程序切片技术及其应用

李必信　编著

国家自然科学基金资助(项目编号:60473065)
计算机软件新技术国家重点实验室(南京大学)资助
东南大学优秀青年教师教学科研资助计划资助
东南大学科技出版基金资助

科学出版社
北　京

内 容 简 介

本书主要介绍程序切片的起源和发展，程序切片技术的图论基础，程序切片的各种变体（如静态切片、动态切片、有条件切片、并发切片、面向对象程序的切片、体系结构切片和规约切片等），计算程序切片的各种方法，以及程序切片技术在软件调试、波动分析、软件测试、度量、重用、程序理解、逆向工程和软件安全等方面的应用，并讨论了程序切片技术的发展趋向。

本书适合作为高等院校计算机软件专业学生学习“软件工程”、“软件分析与测试”等课程的参考书或工具书，也适合作为软件分析与测试研究人员的参考书。

图书在版编目（CIP）数据

程序切片技术及其应用/李必信编著.—北京：科学出版社，2006
ISBN 7-03-016944-1

Ⅰ.程… Ⅱ.李… Ⅲ①软件－分析 ②软件－测试 Ⅳ.TP311.5

中国版本图书馆 CIP 数据核字（2006）第 013265 号

责任编辑：鞠丽娜 刘亚军/责任校对：都 岚
责任印制：吕春珉/封面设计：王 浩

科学出版社 出版
北京东黄城根北街 16 号
邮政编码：100717
http://www.sciencep.com

印刷
科学出版社发行 各地新华书店经销
*
2006 年 3 月第 一 版 开本：B5（720×1000）
2006 年 3 月第一次印刷 印张：19 1/2
印数：1—2 500 字数：390 000

定价：35.00 元
（如有印装质量问题，我社负责调换）
销售部电话 010－62136131 编辑部电话 010－62138978－8002（BI01）

前　言

计算机系统的发展给软件提出了更高的要求，软件质量提升这一课题的研究越来越得到从业人员的重视，而软件的可靠性、正确性、安全性等是保证软件具有高质量的关键因素。近年来，软件从业人员，特别是软件工程的从业人员，为保证开发中软件的质量，提出了多种技术与方法。其中类型推理、代码分析、软件测试、形式验证和 CMM 等就是典型代表。程序切片是由 M.Weiser 先生于 1979 年在他的博士论文中首先建立起来的一种程序分解技术。它通过寻找程序内部的相关性来分解程序，再通过对分解所得程序切片的分析达到对整个程序的分析和理解。

程序切片技术的研究有着极其重要的理论和实际意义。它极大地丰富了程序分析、程序理解、软件维护的理论基础，为软件工程的各个阶段提供理论和技术支持，保证软件开发过程沿着严谨、正确的道路走下去。例如，在需求规约层，我们可以通过对软件形式规约的切片，发现规约中存在的错误，分析产生错误的原因，进而纠正错误；在软件体系结构层，我们可以利用体系结构切片技术，逐步对体系结构规约进行精化（refinement），并验证体系结构的正确性，从而保证按此体系结构设计软件的正确性和可靠性；在设计阶段，我们同样可以利用模块间切片技术，尽可能地设计出高内聚、低耦合的模块，从而降低程序的复杂性，并为软件重用奠定良好的基础；在编码阶段，程序切片技术的应用更加广泛，它在程序分析、理解、测试、调试、度量以及维护过程中都有着极其广泛的用途。作为一种应用广泛的程序分析技术，程序切片正对软件工程领域产生越来越大的影响。

编著本书的目的在于向那些已学过大学“软件工程”、“程序设计语言”等计算机软件基础课程而希望进一步深入学习程序分析方法的读者，提供一部全面介绍程序切片技术的基础读物。本书内容包括程序切片的基本原理和方法，基本工具和技巧，以及程序切片在软件工程各个方面的运用。本书力求以浅显易懂的形式讲述基本理论和方法，并且紧密结合实践。

本书共分上、中、下三篇和附录。

上篇主要阐述程序切片的发展历史和基本原理，它由 9 章构成，分别阐述程序切片基本概况（第 1 章），图论基础（第 2 章），静态程序切片（第 3 章），动态程序切片（第 4 章），条件程序切片（第 5 章），并发程序切片（第 6 章），面向对象程序切片（第 7 章），规约切片（第 8 章），程序切片变体（第 9 章）。

中篇主要阐述程序切片的基本应用，它由 8 章构成，分别论述程序切片在软件调试（第 10 章），波动分析（第 11 章），软件测试（第 12 章），软件维护（第

13 章），软件度量（第 14 章），软件安全（第 15 章），软件重用（第 16 章），应用扩展（第 17 章）等方面的应用现状及热点研究课题。

下篇主要介绍我们在层次切片模型方面的工作，并针对一些新型程序切片课题进行相关讨论，旨在开拓读者的视野。本部分由层次切片模型及其实现（第 18 章）、层次切片模型的应用（第 19 章）和结束语（第 20 章）3 章构成。

附录给出了汉英名词对照和缩略语英汉对照。

作者多年来一直从事程序切片技术研究，并担任东南大学“软件分析与测试”，“软件开发方法与技术”等课程的教学工作。本书主要内容来源于作者的博士论文和相关的研究论文，以及国内外一些著名的相关文献。

作者衷心感谢恩师南京大学的郑国梁教授，师兄南京大学的李宣东教授、陈家骏教授、赵建华教授，以及东南大学的罗军舟教授、徐宝文教授和瞿裕忠教授，是他们的鼓励和无私的帮助激发了我完成本书写作的决心；同时还要感谢我在南京大学学习期间一起做研究的 SEG 同仁，他们是郑滔、樊晓聪、崔萌、梁佳、刘小东、张勇祥、钱雨、沈驿梅、朱平、谭毅、刘芳和陈雨亭等；另外还要感谢东南大学江苏省软件质量研究所的聂长海老师、张迎周老师、史亮同学和章晓芳同学，以及东南大学计算机系 ERICS 组的老师和同学，他们是周颖、张鹏程、王艳臣、莫军辉、苏志勇、杨利利和周宇等。

这里还要特别感谢上海交通大学的赵建军教授，多年来，很多灵感得益于赵教授的论文和与赵教授的多次交流。北京大学的梅宏教授对本书稿提出了许多具有建设性的意见，在此表示衷心的感谢。作者在南京大学学习期间以及在东南大学工作期间均得到了计算机软件新技术国家重点实验室（南京大学）的大力支持。另外，作者在研究程序切片技术的过程中得到了国家 863 基金和国家自然科学基金资助，在此一并表示感谢。

最后，感谢科学出版社的鞠丽娜老师和刘亚军老师，正是她们的关心和支持，才使得本书的写作、出版、发行得以顺利完成。

程序切片技术发展到今天，已经形成多个分支，并积累了丰富的理论研究和经验研究成果。由于时间问题，本书不可能包含所有有关程序切片的内容，感兴趣的读者还需一些辅助资料才能对它有更加深入地理解。由于作者水平有限，错误和遗漏在所难免，恳请读者批评指正。

作　者

2005 年 11 月于东南大学

目　　录

上篇　程序切片技术基本原理

中篇　程序切片技术的基本应用

下篇　程序切片技术展望

附　录

上篇　程序切片技术基本原理

20 世纪 70 年代初期，随着计算机软件技术的发展，特别是计算机软件规模的扩大，相当一部分人正致力于研究较大规模计算机程序的调试和维护问题。在软件调试过程中有人发现：当一个大的计算机程序被分解成一个个较小的程序片段以后，很容易被构造、理解和维护。于是一些用于程序分解的技术和方法被研究人员提出来。例如，信息隐藏、数据抽象和 HIPO 等技术就是其典型代表。这些技术并非互斥，而是互相补充地完成程序分解和抽象的任务。通过对这些技术的反复运用，一些新型技术又不断涌现出来，使得程序分解技术得到进一步的发展和充实，其中程序切片起到了其他技术无可替代的作用，它的出现和发展注定了要在程序分解领域引起一场革命。

程序切片是一种用于分解程序的程序分析技术，它的原理和方法是由 M.Weiser 于 1979 年在他的博士论文中首次建立，后来分别于 1981 年和 1984 年在他的两篇论文中又得到了进一步的完善和推广。M.Weiser 博士不仅建立了程序切片的概念，并提出了基于控制流图的计算程序切片的算法，以及基于程序切片的软件度量基本框架等，为后来人们扩展程序切片及其应用奠定了坚实的基础。程序切片一经问世，就引起了软件从业人员极大的兴趣，很多研究者们开始进行这一课题的研究，经过 20 多年的努力，程序切片技术理论日趋完善，已经形成了多个分支，应用范围几乎遍及软件工程学科的各个方面。纵观全局，程序切片经历了以下几个发展阶段。

（1）基于数据流方程的程序切片阶段

这一阶段主要是指由 M.Weiser 提出的程序切片概念，以及运用基于 CFG 的数据流方程来计算程序切片的算法。

（2）基于依赖图的程序切片阶段

这一阶段主要包括：K.J.Ottenstein 等人分别于 1984 年和 1987 年引入了基于程序依赖图的图可达性算法，它能够计算当时流行的过程内后向切片；S.Horwitz 等人分别于 1988 年、1990 年和 1992 年引入了前向切片的概念及算法，过程间切片概念以及基于系统依赖图的两步遍历图可达性算法；B.Korel 和 J.Laski 于 1988 年，H.Agrawal 和 J.R.Horgan 于 1990 年分别引入了动态切片的概念和算法。

（3）面向对象程序切片阶段

随着面向对象程序设计语言和设计方法的兴起，人们自 1996 年开始了面向

对象的程序切片技术的研究。其中，M.J.Horrald 等人利用类依赖图来扩充传统的系统依赖图使其能够表示面向对象的 C++程序，然后利用改进的两步遍历图可达性算法来计算 C++程序的程序切片；C.Steindl 通过对各种各样的控制流和数据流的计算，并结合他们所使用的一个项目语言来讨论面向对象程序的切片计算问题；另外，J.Zhao 教授，D.Liang 博士和 Z.Chen 博士等人分别在面向对象动态程序切片、使用对象依赖图切片面向对象程序，以及面向对象 Java 程序切片等方面各自提出了自己的观点。

（4）程序切片发展“百花齐放”阶段

这一阶段的主要标志就是出现了各种各样的程序切片变体，如削片、砍片，还有数据切片、层次切片和无定型切片等。同时，程序切片技术的应用也在此阶段得到了极大的丰富和发展。除了传统的基于切片的软件调试、软件测试和程序理解以外，程序切片还被广泛运用于程序重组、逆向工程、软件维护、程序验证以及软件可靠性分析和建模等方面。

以上四个阶段的划分没有严格的时间限制，甚至前后阶段之间存在着较大的时间重叠，笔者只是为了问题叙述的方便，仅供读者参考。另外，除了这里提到的作者以外，还有一大批从事程序切片研究并做出突出贡献的同仁，他们的工作也是极具理论和应用价值的。例如，有关并发程序切片，有关有条件切片等，我们将用独立的章节分析和介绍这些成果。

本篇主要是对程序切片技术发展的一个回顾，所包含内容尽量避免深奥的理论赘述，相信初学者可以比较轻松地学习每一个章节，同时又能够较快地把握整个技术发展的轮廓。

本篇的主要内容组织如下。

第 1 章介绍程序切片技术的发展、基本原理和应用，以帮助读者快速入门。

第 2 章介绍控制流图、程序依赖图、控制流分析和数据流分析的基本原理。

第 3 章主要介绍 M.Weiser 程序切片的基本思想，过程内切片和过程间切片等，目的是让读者了解程序切片发展早期阶段的有关情况。

第 4 章详细论述动态程序切片的基本原理，讨论计算动态切片的基本方法。

第 5 章简述有条件切片的基本思想及其应用。

第 6 章讨论并发程序的切片问题。

第 7 章从面向对象程序的特点出发，论述计算面向对象程序切片时遇到的问题和挑战，以及目前采用的计算面向对象程序切片的方法等。

第 8 章重点阐述针对规约语言的程序切片思想和相应的算法等。

第 9 章讨论的是一些新型的程序切片问题，对其中的许多问题进行初步的论述和探讨。

第 1 章　概　　论

本章比较全面地概述了程序切片技术的起源和发展现状，把程序切片技术的发展概括为：从基于数据流方程的算法到基于依赖图的算法，从可执行的程序切片到不可执行的程序切片，从静态切片到动态切片再到有条件切片，从后向切片到前向切片，从代码切片到规约切片，从单一切片到多种切片的过程。内容还包括：切片准则的变化和发展，程序切片技术的应用现状，切片工具的研制和使用等。旨在让读者对这一热门技术有一个概括的了解，为阅读后续章节提供线索。

1.1　程序切片技术的起源和发展

程序切片的基本思想由 M.Weiser 于 1979 年在他的博士论文中首次建立。1981 年他又在国际软件工程大会（ICSE'81）上报告了这一新发现，立即引起了当时计算机界不小的震动，他提交给大会的论文《程序切片》(Program Slicing)获得当届“今后十年最佳论文奖”。当时的情况是，M.Weiser 等人正在进行自动程序抽象的理论和实践研究，他们通过对源程序代码的分析以及对控制流图(control flow graph，CFG）的研究发现：程序的某一个输出只与源程序中部分语句和控制谓词（control predicate）有关，删除其他的语句和谓词并不影响该输出的结果。这表明：对该输出来说，源程序与删除不相关语句和谓词以后所得的可执行程序在语义上是一致的。M.Weiser 等人把这种只与某个输出有关的语句和谓词构成的程序称溪源程序的一种静态切片（static slice），并提出了基于 CFG 的计算程序切片的算法。1984 年，M.Weiser 博士在国际权威杂志 IEEE Transactions on Software Engineering 上发表了同样题为《程序切片》的论文，进一步形式地阐述了程序切片技术的基本思想，以及如何基于数据流方程计算程序切片的方法。从此，越来越多的专家学者开始了这一课题的理论和应用研究。如今，程序切片技术的研究和发展已经经历了 20 多年的漫长旅程，在理论和应用两个方面都取得了丰硕的成果，这些研究成果在计算机学科的很多方面发挥了应有的作用。而且从各方面来看，程序切片技术还会作为一种主要手段在软件开发的各个阶段、程序设计语言、硬件描述语言、其他规约语言和形式化模型等的分析和测试方面继续发挥不可低估的作用。笔者根据多年从事程序切片技术的研究体会，从程序切片的算法、切片准则的演化和切片变体的形成、切片技术的应用以及切片工具等方面简要介绍程序切片技术的发展历程。

1.1.1 从数据流方程到程序依赖图

M.Weiser 提出程序切片的基本概念以后，接下来的问题是如何计算程序切片。于是，M.Weiser 又在当时比较流行的程序控制流图的基础上建立了数据流方程，然后，通过求解数据流方程来获得相应的程序切片（详细过程参见第 3 章）。但是这种计算程序切片的算法明显地存在一些问题。例如，计算程序切片的效率不高，因为为了计算程序切片，往往要求解一系列迭代方程，所消耗的时间和空间均比较多；计算所得的程序切片准确性也不是很好，这主要是因为，建立数据流方程的过程是一个非常复杂的过程，在这个复杂的方程建立过程中，很容易犯这样或那样的错误，有时即使是一些很小的错误信息也会导致结果切片的不准确；另外，基于数据流方程求解程序切片算法的应用也受到很大的限制，这是因为 M.Weiser 提出这种算法的时候，主流程序都是一些只包含基本结构（顺序、选择和循环结构）的程序，对这些程序建立数据流方程基本上问题不大，但是随着面向过程的编程语言的兴起和广泛应用，如何把程序切片技术运用到过程程序中就显得特别重要，这时利用基于数据流方程求解程序切片的方法几乎不可行，因为对带有过程（特别是多个过程）的程序建立数据流方程已经非常困难，从某种意义（如效率和准确性）上说已经毫无意义。为此，寻找新的计算过程程序切片的方法成为当时研究的热门课题。

“道高一尺，魔高一丈”，人类的智慧是无限的。就在 1984 年，K.J.Ottenstein 和 L.M.Ottenstein 发表了题为《软件开发环境中的程序依赖图》（The Program Dependence Graphin a Software Development Environment）的研究论文，从而使人们从传统的控制流图中摆脱出来。程序依赖图（program dependence graph，PDG）是根据程序中存在的各种依赖关系抽象出来的一种程序中间表示，它的节点表示程序语句或控制谓词，边表示依赖关系。而在 CFG 中，节点表示程序的基本模块（basic block），边表示控制流。

1987 年，J.Ferrante，K.J.Ottenstein 和 J.D.Warren 三位学者联合在国际一流杂志 ACM Transactions on Programming Languages and Systems 上发表了题为《程序依赖图及其在优化过程中的运用》（The Program Dependence Graph and Its Use in Optimization）的论文，首次提出了基于程序依赖图的图可达性算法，从而解决了当时非常流行的过程内后向切片（intraprocedural backword slicing）的计算问题。

但是，这种基于 PDG 的图可达性算法只能计算一个过程内的后向切片问题，含有多个过程的程序切片计算问题并没有得到解决。于是，学者们继续艰苦的探索过程，直到 1988 年，由 S.B.Horwitz，T.W.Reps 和 D.Binkley 三位教授联合发表的论文中首次提出了利用系统依赖图（system dependence graph，SDG）计

算过程间程序切片的两步遍历图可达性算法（two-pass graph reachability）。从而解决了包含多个过程程序的切片计算问题。

为此，程序切片计算算法完成了从基于控制流图的数据流的方法到基于依赖图的图可达性方法的转变。这一转变使程序切片的计算手段发生了质的变化，它把程序切片的计算过程变得简单而且可视化。基于程序依赖图计算切片的过程大致如下：

1）构造某个程序的 PDG 或 SDG；

2）给定某个程序切片准则，在 PDG 上运用图可达性算法，或在 SDG 上运用两步遍历图可达性算法，获得与切片相对应的依赖图（我们称之为依赖图切片）；

3）把依赖图切片映射到源程序中，与依赖图切片相对应的语句和控制谓词构成的集合就构成了该程序关于此切片准则的程序切片。

1.1.2 从可执行的程序切片到不可执行的程序切片

程序切片在 20 多年的发展过程中经过了一系列的演化，甚至连“程序切片”的定义（内涵和外延）也都发生了变化。最早的程序切片是由 M.Weiser 博士定义的，他认为，程序切片是一个可执行的程序，切片与源程序对某个特定的变量（包含在切片准则中的变量）来说，在程序的某个特定地方语义上应是一致的。

一般来说，把满足下列两个条件的程序切片称为 M.Weiser 程序切片：

1）一个程序切片对应一个特定的切片准则（slicing criterion），记为〈n,V〉，其中 V 表示在 n 定义或使用变量的集合，n 指程序中的某个点（一般指某条语句）；

2）程序 P 的切片 S 可通过从 P 中删除零条或多条语句得到，但要保证程序 P 和切片 S 关于切片准则〈n,V〉的行为相同（behave the same）。

在图 1.1 和图 1.2 中，可以看到有语法和语义两方面的问题：根据语法，从 P 中删除多余语句可以得到 S。根据语义，对所有的初始状态，P 和 S 对第 10 句的 x 都会导致相同的终值。可见，M.Weiser 的程序切片是一种可执行的程序。

后来的研究者们推广了 M.Weiser 博士的定义，它们把一些不可执行的程序也包含进来，从而丰富了程序切片的内涵。例如，H.Agrawal 和 J.R.Horgan 两位学者于 1988 年提出的动态切片，以及 S.B.Horwitz，T.W.Reps 和 D.Binkley 提出的过程间程序切片的概念，在某种意义上都不必是一种可执行的程序。它们定义程序切片的典型方法就是：给定某个程序 P，程序切片准则〈n,V〉，程序 P 关于切片准则〈n,V〉的程序切片（如静态后向切片）是由程序 P 中所有在 n 点对变量 v∈V 的值有影响的语句和控制谓词构成。显然，通过这种方式定义的程序切片不一定是可执行的。

```
1    begin
2        x:=y
3        if x=3
4        then
5        begin
6            c:=y
7            x:=25
8        end
9        i=i+1
10   end
```

图 1.1 程序 P

```
1    begin
2        x:=y
3        if x=3
4        then
5         *****
6             ****
7            x:=25
8         ***
9         *****
10   end
```

图 1.2 程序 P 的切片 S

1.1.3 静态切片、动态切片和有条件切片

根据前面小节的讨论知道，程序切片的定义一般包含下面两个部分。

1）切片准则 sc 代表感兴趣的变量以及感兴趣的变量的定义位置或使用位置，一般是一个二元组〈n，V〉，其中 n 表示程序 P 中的某个兴趣点（如某条语句）；V 代表程序 P 中在 n 点定义或使用的变量（或变量的集合），根据不同的切片准则，可以得到不同的切片。

2）程序 P 中与切片准则 sc 相关部分的集合 S（一般来说）就是我们要找的关于切片准则 sc 的程序切片。

这里有一个值得关注的问题，那就是在计算这类程序切片的时候，并没有考虑程序的具体输入是什么。在计算程序切片时，如果要考虑程序的具体输入，就引出了程序切片的另外两个变种，即动态切片（dynamic slice）和有条件切片（conditional slice）。其中动态切片由 B.Korel 和 J.Laski 最早提出，它与程序的某个特定的输入 I_0 有关。动态切片准则是一个三元组〈n，V，I_0〉，要计算程序 P 的动态切片就是要计算程序 P 中在某个特定输入 I_0 的情况下所有影响 V 在 n 点的值的语句和谓词组成的集合。有条件切片的提出最早可见 G.Canfora，A.Cimitile 和 A.D.Lucia 三位学者于 1998 年在国际著名杂志 Information and Software Technology 上发表的学术论文《有条件的程序切片》（Conditioned Program Slicing）。在该文中，他们首次建立了如何计算有条件程序切片的思想。从有条件切片的定义来看，它是介于静态切片和动态切片之间的一种切片形式：因为利用有条件切片技术计算切片时考虑某些满足某个谓词表达式的所有可能输入（假设是 I_c），利用静态切片技术计算切片时要把程序的各种可能的输入都考虑进去（假设是 I），而利用动态切片技术计算切片时只考虑某个特定的具体输入。很显然它们之间存在下列关系：$I_c \subseteq I$，$I_0 \in I$。

从刚才的讨论中，我们不难发现：程序切片的演化与切片准则有关，因为如果切片准则设置为〈n,V,I_0〉，我们计算所得到的程序切片就是动态切片；如果切片准则设置为〈n,V,I_c〉，则我们计算所得到的程序切片就是有条件切片；如果切片准则设置为〈n,V〉，很显然就得到静态切片了。

1.1.4 后向切片和前向切片

前面提到的所有切片都是后向切片（backward slice），因为它们要寻找的都是程序 P 中那些对某个兴趣变量有影响的语句和控制谓词，也就是要构造集合 affect（V/n），该集合由所有影响 V 在 n 点值的语句和控制谓词构成。细心的读者可能会问：如果我们想知道程序 P 中究竟有哪些语句或谓词受到 n 点的变量 V 的值的影响，那该如何计算？我们的回答是只要计算前向切片（forward slice）即可。要计算前向切片实际上就是构造集合 affected-by（V/n），该集合由程序 P 中所有受到变量 V 在 n 点的值影响的语句和控制谓词构成。

在程序分析和测试过程中，前向切片和后向切片的作用均不可忽视。例如，如果对一个已经测试完成的程序进行了简单的修改，我们一定要知道这点修改可能会带来哪些副作用，也就是这点修改可能会影响后面哪些语句和控制谓词。显然这是一个只要计算前向切片的问题。而另一方面，如果我们在程序测试过程中发现了某个错误，我们一定要知道是谁制造了这个错误，也就是前面的哪条语句或哪个谓词表达式产生了这个错误，并如何传播到我们发现它的地方。这显然是一个计算后向切片的问题。

1.1.5 从源程序代码切片到软件规约切片

尽管程序切片来源于程序调试过程，而且早期的应用也主要集中在程序分析、程序理解和维护，以及软件回归测试等领域。但是程序切片技术可以适用软件开发各个阶段的文档（如需求规约、设计规约、代码和字节码等），而且可以用于各种文档的分析、理解、测试、维护、信息提取和安全分析等。目前程序切片技术作为一种主要的手段被广泛用来解决软件质量保证过程中的一系列关键问题。

虽然人们对程序切片技术进行了长达 20 多年的研究，但对规约切片（specification slice）的研究起步却比较晚，很多研究还有待于进一步深入。1993 年日本学者 T.ODA 和 K.ARAKI 首次把程序切片技术运用于软件开发的形式化软件规约中，给出了形式规约切片的定义，提出了在二元依赖关系和多个表达式集的基础上计算静态形式规约切片的方法，并尝试着运用到调试中。但是，该算法仅仅考虑数据依赖，因此得到的切片结果精确度低，而且也存在着诸如部分标识的依赖域由用户自己定义的缺陷。

随后，J.Chang 和 D.J.Richardson 于 1994 年在此基础上提出了动态形式规约切片的概念，进而把形式规约切片划分为静态规约切片和动态规约切片。静态形式规约切片包含了可能限制变量值的形式规约，是针对后置条件中的变量而言的，它可运用到区分规约的不同部分中去；动态形式规约切片包含了在实际的操作序列中真正影响变量值的形式规约，相比静态形式规约切片而言，动态形式规约切片是一项更有效的技术，因为动态形式规约切片比相应的静态形式规约切片更小，可运用到规约的有效性检验以及规约的调试和测试等方面。另外，作者还把依赖关系细分为数据依赖和控制依赖，指出控制依赖不仅可以发生在模式内，也可以发生在模式间，缺陷是忽略了模式包含和模式作为类型出现时的控制依赖情形，也忽略了通用模式（参数传递）、公理、递归定义（自由类型的定义、如何描述队列和树这样的结构）等的数据依赖和控制依赖的情形，只是给出了控制依赖和数据依赖的定义，并没有给出计算方法。

目前，一些研究者还对状态机切片进行了探讨，如 B.Korel 等人在 EFSM (extended finite state machine) 的基础上以迁移为分析单元提出了基于 EFSM 的切片方法，但在建立迁移依赖图时，存在迁移不能定义或需要修改多个变量等缺陷，并且忽略了 EFSM 中的重要组成部分——状态。

程序切片技术在软件规约的应用，部分解决了当时正在兴起的模型检验 (model checking) 过程中遇到的状态空间爆炸问题。同时由于软件规约在软件开发过程中的核心作用，软件规约的正确性是保证软件开发向着正确、有效方向发展的重要里程碑。1998 年旅日中国学者 J.Zhao 博士首次把程序切片技术运用到软件体系结构中，从而引起了对体系结构切片的广泛关注。1999 年，旅美学者 T.Kim，Y.T.Song，L.Chung 和 D.T.Huynh 又把动态程序切片引入到体系结构规约中。目前基于状态机规约的切片和基于 UML 模型规约的切片及其研究仍然是一个很热门的课题。

1.1.6 从单一切片到多种切片

目前，程序切片的变体比较多，五花八门，容易引起初学人员的混淆。笔者认为：一般情况下，事物的发展是一个顺序演进的过程，本质上应该是一致的。如果要发生根本变化，只有变革。尽管程序切片从早期的静态后向切片发展到现在的静态前向切片、动态（前向和后向）切片、有条件切片、组合切片、无定型切片、削片、砍片等。其实都不是什么本质的变化。所有这些变化不过是程序切片准则的演化和变体。静态后向切片的切片准则是一个三元组〈n，V，backward〉，其中 n 是程序 P 中的某个语句号，V 是在语句 n 点定义或使用的变量和变量集合，而 backward 表示切片方向（这里是后向）。由此切片准则，我们很容易知道，我们要计算的就是“程序 P 中所有对变量（或变量集）V 在 n 点的值有影响

的语句和控制谓词的集合”（称为“静态后向切片”）。如果切片准则改为〈n,V,forward〉，则我们要计算的就是“程序 P 中所有受到变量（或变量集）V 在 n 点的值影响的语句和控制谓词的集合”（称为“静态前向切片”）。更进一步，当切片准则变成〈n,V,I〉时，其中 I 表示程序的可能输入集合（或可能输入集合的某个子集，或某个特定输入）时，我们很容易理解它们分别表示静态切片、准静态切片和动态切片了。随着程序切片准则的变化，我们可以得到不同的程序切片类型，同时可以定义不同领域、不同层次的切片。例如，如果用〈n,V,P〉表示程序 P 的某个切片准则，则可以用〈n,V,S〉表示规约层次的切片准则，然后再根据类似的演化过程，得到软件规约的不同类型的切片变体。当然，也有一些切片变体的定义比较复杂，不只是如此简单的演化，如无定型切片、削片和砍片等，但是有一点可以肯定，它们都是在切片应用的驱动下产生的切片变体，代表了切片技术的一类发展方向，本书将用专门章节讨论这类切片。

1.2 程序切片技术的应用概述

程序切片技术在软件分析、理解、调试、测试、度量、软件质量保证、逆向工程等许多方面有着广泛的应用，本节简单介绍其在程序调试、程序集成、软件维护、回归测试、软件度量，软件重用，以及软件安全中的应用。

1.2.1 程序调试

在软件调试过程中，定位程序中存在的错误是一项困难的工作，程序切片可以帮助程序员很容易地进行错误定位。若检测到程序点 S 上的某个变量 V 存在错误，我们可以把〈S,V〉作为切片准则来计算后向切片，引起错误的代码就限定在这个切片中，通常情况下，这会大大降低要分析的代码量。

为了进一步帮助程序员定位错误，人们开发了几种程序切片的变种：程序削片（program dicing）和程序砍片（program chopping）。程序削片是利用一些没有通过测试和已经通过测试的变量的切片来分析程序中哪些语句可能存在错误。这些切片的不同方式的组合可以得出一些有意义的结论：两个切片的交集应该包含导致两个切片都出现错误的语句；切片 S_1 与 S_2 补集的交集可以从 S_1 中排除不会导致第二个测试用例错误的所有语句。

程序砍片可用于识别位于程序点 a 与 b 之间、受 a 的修改影响的语句。这对于分析由于程序中某个位置的修改而导致另一个位置程序错误的情况非常有用。

1.2.2 软件维护

软件维护是软件生命周期的重要组成部分，其费用占整个系统费用的一半以

上。软件维护的主要挑战来自对现有系统的理解和确保修改系统不会引发新的错误。

分解切片（decomposition slice）捕获与程序中某个变量在所有计算位置的值相关的程序信息，将对程序的修改引起的影响控制在一定的范围内，而不会带来无法预料的副作用。它需要计算程序中某个变量在所有位置的切片，也需要计算所有变量的分解切片。变量 v 的分解切片是与变量 v 有关的关键节点切片的并集，其中，关键点是指输出变量 v 的节点和程序的出口节点。这样，对一个变量 v 而言，一个程序就被划分为三个部分。如果用矩形表示程序 P，椭圆 E_a 表示与 a 相关的分解切片，椭圆 E_b 表示与 b 相关的分解切片，椭圆 E_c 表示与 c 相关的分解切片，则一个分解切片把程序分解三个部分：独立部分、相关部分和补集部分。

（1）独立部分

独立部分（independent part）包含所有只被 v 的分解切片包含的语句。私有部分的语句可被删除而不会影响其他部分。若对变量 v 的所有赋值都集中在私有部分，则变量 v 为可修改（changeable）变量。可以在切片的任何位置中添加对可改变变量的赋值语句。

（2）相关部分

相关部分（dependent part）包含所有既在 v 的分解切片中又在其他变量的分解切片中的语句。若至少有一个对变量 v 的赋值在公共部分，则变量 v 为不可修改（unchangeable）变量。修改这种变量将会引起其他部分的改变。

（3）补集部分

补集部分（complement）包含所有不包含在 v 的分解切片中的语句。修改切片中的语句时，补充部分应该保持不变。

在修改代码时，只需要关注私有部分和公共部分，修改后只有这两部分需要重新测试，补集部分不会被改变，因此不需要程序测试。$E_a-(E_a\cap E_b)\cup(E_a\cap E_c)$，$E_b-(E_b\cap E_a)\cup(E_b\cap E_c)$和 $E_c-(E_c\cap E_a)\cup(E_c\cap E_b)$都是各自分解切片形成的独立部分，$E_a\cap E_b$，$E_b\cap E_c$，$E_a\cap E_c$，$E_a\cap E_b\cap E_c$ 是相关部分，而 $P-E_a\cup E_b\cup E_c$ 是补集部分。

1.2.3 回归测试

当软件系统修改后，为了保证系统的正确性以及修改不会带来副作用，需要对其进行重新测试，即回归测试。回归测试不同于软件开发过程中的测试，因为已经存在一些测试用例供选择（当然，必要时还需要设计新的测试用例）。对于这些测试用例，我们可以全部重新测试一遍或者选择一部分进行测试。完全重新测试可能会消耗大量的资源，并且大部分情况下只有一小部分代码会受到修改的

影响，因此，只有那些受影响的代码需要重新测试。对通过分解切片获得的补充部分便不需要重新测试。程序切片技术可以帮助我们获得那些受影响的代码，从而有助于选择测试用例，进一步分析需要测试的代码、简化测试过程等。

1.2.4 软件度量

软件度量的两个重要度量指标就是内聚（cohesion）和耦合（coupling）。其中内聚度指模块内部各成分之间的关联程度。模块的内聚度越高，越容易理解、修改和维护。为了度量模块的内聚度，L.M.Ott 和 J.J.Thuss 提出了数据切片（data slice）的概念，它计算关于数据标记（data tokens）而不是语句的切片，其中数据标记是指变量和常量的定义和使用。关于数据标记 v 的数据切片包含：v 的切片集合中的语句所包含的数据标记序列。

需要对每个输出计算一个数据切片，一个输出是指输出到一个文件、变量赋值、输出参数、对全局变量的赋值等等。若一个数据标记存在于多个数据切片中，则称这个标记为胶水（glue）标记。通过胶水标记可以将不同的数据切片连接起来。若一个数据标记存在于所有数据切片中，则称这个标记为强力胶水（super-glue）标记。标记的粘性（adhesiveness）用来度量与一个标记连接的切片数目。强功能内聚度（strong functional cohesion）定义为“强力胶水标记的个数与切片中数据标记的总数的比值”；弱功能内聚度（weak functional cohesion）定义为“胶水标记的个数和切片中数据标记的总数的比值”。根据这些定义，就可比较直观地分析模块的内聚度。

M.Harman 首次把程序切片技术运用到模块间的耦合度量中，后来笔者进一步把他的方法推广到面向对象的程序中，并基于层次切片模型建立了一个用于度量 Java 程序耦合的框架。根据框架，我们可以计算 Java 程序不同层次组件之间的耦合度，如包之间、对象之间、方法之间、语句之间，甚至是变量之间的耦合，还可以计算某个具体组件自身的耦合度。

1.2.5 软件重用

J.Zhao 和 M.J.Horrald 等人分别从不同的层次讨论过基于切片的可重用组件的提取问题。J.Zhao 利用软件体系结构切片提取可重用的软件组件和连接件，甚至是配置，而 M.J.Horrald 等人从代码层次利用程序切片提取相应的可重用代码。

1.2.6 软件安全

尽管软件安全看起来似乎和程序切片没有多大的关系，但是利用程序切片技术仍然可以解决一些与软件安全确认相关的关键问题。美国学者 K.B.Gallagher

和 J.R.Lyle 就是利用程序切片解决软件安全问题的开创人员。他们利用切片技术来分析和控制临界安全组件（critical-safety component）的传播问题。还有一些学者利用程序切片技术分析和解决程序执行过程中的信息泄漏等软件安全（security）问题。

实际上，程序切片的应用非常广泛，在软件开发的各个阶段都可以发挥程序切片的巨大作用，我们将在本书的后续相关章节详细介绍。

1.3 程序切片工具简介

目前，实用的程序切片工具（program slicing tool，PST）和环境很多，按照切片对象的程序设计语言的不同可分为以下几类：

1.3.1 支持 C 语言的 PST

Wisconsin 程序切片工具 Version 1.1 是一个支持对 C 程序操作的软件系统。它包括后向切片、前向切片和消片三种功能，他们能够帮助用户获得对一个程序正在做什么和如何做的理解。

ChopShop 是由 Carnegie Mellon 大学的计算机科学学院最初开发的一种逆向工程工具，可用来帮助程序员理解不熟悉的 C 代码。其中采用了一种新的数据流分析技术，即静态程序切片的模块产生机制。它给出一种比标准形式的切片更易理解的结果。用户能够选择几种信息源和信息池（sinks of information）。Chopshop 显示数据是如何从源流到池。Chopshop 接受完全的 ANSI，以文本和图形两种形式产生程序切片，并进行化名分析、处理所有的语言特性，且处理的程序比当时相应的系统都大。它是第一个为过程提供数据抽象机制的程序切片工具。

Ghinsu 是由 Florida 大学计算机和信息科学系开发，受到软件工程研究中心（SERC）基金资助的一个集成了许多工具的程序切片环境，可以在软件工程的许多活动中，主要是维护活动过程中起到辅助作用。当前版本的 Ghinsu 要求的输入必须是一个用 C 的一个大子集（包括指针）书写的一个程序。

Unravel 是一个原型程序切片工具，可以用来使用程序切片静态地估价 ANSI C 源代码。该项目的开发得到美国核控制委员会（NRC）和国家通信系统（NCS）的基金资助。通过组合程序切片和逻辑集合操作，Unravel 能够识别在不止一次计算中执行的代码。这种信息对阐述高级集成软件及时有用，因为包含在这种代码中的一次失败可以导致多个逻辑软件构件的故障。

1.3.2 支持 Ada 语言的 PST

PSS/Ada 系统是由杨洪等人设计的一个 Ada 程序的静态切片生成原型系统。它以对 Ada 程序进行静态程序依赖性分析（包括数据依赖分析和控制依赖分析）为基础，分析了 Ada 程序中语句间存在的波动效应（ripple effect），并生成特殊的 Ada 程序切片，该切片是从 Ada 程序中抽取出的某些语句以相同的顺序重新组成的一个新程序段。PSS/ Ada 系统是一个 Ada 软件的测试、排错、分析、理解和维护工具，在 Ada 程序的并行性检查、波动性分析、复杂性度量等方面也提供一定范围的支持。PSS/ Ada 系统作为对 Ada 程序设计支持环境 APSE 的工具集的扩充，丰富了开发 Ada 软件的环境。

1.3.3 支持 Oberon-2 语言的 PST

The Oberon Slicing Tool（OST）是由 C.Steindl 开发的一种计算 Oberon-2 程序的程序切片工具，对程序没有限制，也就是说，程序可以使用结构化类型（记录和数据）、任何类型的全局变量、在堆上的对象；过程可以嵌套；由于类型绑定过程（或方法）和过程变量（或函数指针）的存在，程序可以是递归（recursion）的和动态联编（dynamic binding）的；程序也可以由模块组成。OST 切片的速度是相当快的（几秒钟之内就可计算切片信息，一点也不耽误计算切片的时间）。OST 具有下列主要特点：

1）OST 使用静态切片(能产生比动态切片更大的切片且效率也更高)；

2）可以计算过程内、过程间和模块间切片；

3）既可以处理过程程序，也可以处理面向对象程序；

4）OST 是以面向表达式的方式来计算切片(比面向语句的切片粒度更细)；

5）使用用户反馈来限制化名和动态联编的影响；

6）使用一个仓库来储存已计算的切片信息，当输入已经切片的模块时要重用这些信息；

7）实现了模型－视图－控制器（MVC）概念。

1.3.4 支持 Java 语言的 PST

美国堪萨斯州州立大学的 V. P. Ranganath 小组开发的 Java 程序切片器——Indus，是一个比较完整的 Java 程序切片工具，当然 Indus 是基于传统的面向对象程序切片技术开发出来的，具有和其他语言的程序切片工具类似的特性和缺点。1998～2001 年期间，笔者所在南京大学程序切片研究小组在总结前人研究经验的基础上提出了一种层次切片模型，使得可以以一种逐步求精的方式来计算 Java 程序的切片问题。目前，我们开发的切片工具 Jato 已经用于 OOPQL 语言环

境中提供相关程序的切片查询服务。

1.3.5 其他 PST

GrammaTech 是为商业软件开发者、政府机构、著名大学提供的一种创新的软件开发工具。GrammaTech 产品是基于语言的，这意味着它需要理解特定程序设计语言的规则和结构。这使得 GrammaTech 工具可以使许多任务自动化（而大多数工程师还在用传统的基于文本的工具来手工完成这些任务）。

CodeSurfer 以一种新的方式来分析、理解和浏览源代码。CodeSurfer 是一个具有突破性的软件开发和维护工具，工程师们用来浏览和理解源代码中详细的依赖关系。CodeSurfer 提供了一种比较容易访问程序深层结构的方法，即通过全局程序分析发现语义特性和关系。传统工具甚至连看都看不到。

另外，The Linz Oberon Slicing System 是 Johannes Kepler 大学的系统软件组开发的基于 Oberon 语言的模块间切片工具。Microsoft 公司的 slicing tool 是 Microsoft Research 程序分析组开发的一种程序切片工具。Spyder 是由 Purdue 大学计算机科学系开发的一种切片工具，它包含软件调试的高级方法：有界反跟踪新方法的解释、动态切片、切片启发式以及主动调试评估等。

小 结

自从 1979 年 M.Weiser 首次提出程序切片概念以来，程序切片技术的研究和发展已经经历了 20 多个春秋。在这 20 多年中，来自全世界的许多学者都置身于这一课题的研究和开发，取得了大量的具有理论和应用价值的成果。本章从程序切片变体的形成、应用领域和范围的扩展、原型工具的开发等方面比较详细地介绍了程序切片技术的最新进展。旨在让初学者一开始就能对这一热门技术有一个全面了解，并为后续章节的学习打下一定的基础。

思 考 题

1. 你从程序切片技术的发展过程中发现了哪些潜在的规律？

2. 程序切片是一种程序分解技术，你能列举一些其他的程序分解技术吗？它们之间有哪些异同点？

3. 除了本章所列举的几种程序切片技术的应用之外，你还能举例说明一些其他应用吗？

参 考 文 献

陈振强. 2003. 基于依赖的程序切片研究. 东南大学博士论文

李必信. 2000. 面向对象程序切片技术及其在软件度量和软件测试中的运用. 南京大学博士论文

李必信, 郑国梁, 王云峰. 2000. 一种分析和理解程序的方法——程序切片. 计算机研究与发展. 37 (3): 284～291

杨洪, 徐宝文. 1997. PSS/Ada 程序切片系统的设计和实现. 计算机研究与发展, 34 (3): 217～222

张勇翔, 李必信, 郑国梁. 程序切片技术及其应用. 计算机科学. 2000, 27 (1): 31～35

Agrawal H, Horgan J R. 1990. Dynamic program slicing. In: Proceedings of the ACM SIGPLAN' 90 Conference on Programming Language Design and Implementation. 246～256

Harman M, Danicic S. 1997. Amorphous program slicing. In: Proceedings of the 5th IEEE International Workshop on Program Comprenhesion (IWPC' 97). 70～79

Beszédes A, et al. 2002. Union slices for program maintenance. In: Proceedings of International Conference on Software Maintenance (ICSM' 02). 12～21

Binkley D. 1992. Using semantic differencing to reduce the cost of regression testing. In: Proceedings of International Conference on Software Maintenance. 41～50

Canfora G, CimitileA, et al. 1998. Conditioned program slicing. Information and Software Technology, 40 (11～12): 595～607

Chen Z, Xu B. 2001. Slicing concurrent Java programs. ACM SIGPLAN Notices. 36 (4): 41～47

Chen Z, Xu B. 2001. Slicing object-oriented java programs. ACM SIGPLAN Notices. 36 (4): 33～40

Gallagher K B, Lyle J R. 1991. Using program slicing in software maintenance. IEEE Transactions on Software Engineering. 17 (8): 751～761

Harman M, Danicic S. 1995. Using program slicing to simplify testing. Software Testing, Verification and Reliability. 5 (3): 143～162

Hatcliff J, et al. 2000. Slicing software for model construction. Higher-order and Symbolic Computation. 13 (4): 315～353

Horwitz S B, et al. 1988. Interprocedural slicing using dependence graphs. ACM SIGPLAN Notices. 23 (7): 35～46

Jackson D, Rollins E J. 1994. A new model of program dependences for reverse engineering. In: Proceedings of the 2nd ACM SIGSOFT Symposium on Foundations of Software Engineering. 2～10

Korel B, Laski J. 1988. Dynamic program slicing. Information Processing Letters. 29 (3): 155～163

Krinke J. 1998. Static slicing of threaded programs. ACM SIGPLAN Notices 33 (7): 35～42

Larsen L, Harrold M J. 1996. Slicing object-oriented software. In: Proceedings of the 18th

International Conference on Software Engineering. 495～505

Ott L M, Thuss J J. 1993. Slice based metrics for estimating cohesion. In: Proceedings of IEEE-CS International Metrics Symposium. 71～81

Ottenstein K J, Ottenstein L M. 1984. The program dependence graph in a software development environment. ACM SIGPLAN Notices. 19 (5): 177～184

Ricca F, Tonella P. 2001 Web application slicing. In: Proceedings of International Conference on Software Maintenance. 148～157

Steindl C. 1999. Program slicing for object-oriented programming languages. PhD thesis, Johannes Kepler University Linz

Weiser M. 1984. Program slicing. IEEE Transactions on Software Engineering, 10 (4): 352～357

Zhao J. 1998. Applying slicing technique to software architectures. In: Proceedings of the 4th IEEE International Conference on Engineering of Complex Computer Systems. 87～98

Zhao J. 1999. Slicing concurrent Java programs. In: Proceedings of the 7th IEEE International Workshop on Program Comprehension. 126～133

第 2 章　图 论 基 础

在本章中，我们简要介绍与程序切片相关的图论基础部分：控制流图和程序依赖图，为以后介绍程序切片的相关章节奠定理论基础。本章的主要内容组织如下：第一节介绍控制流图的一些基本概念和基本特性；第二节是对控制流分析的简单论述，其中支配节点问题是主要论述内容；第三节讨论数据流分析问题，其中数据流方程的建立以及活性分析是重点讨论对象；第四节展示的控制依赖和数据依赖概念是以后建立依赖图的基础；第五节重点阐述程序依赖图的简单思想，其中过程依赖图和系统依赖图在以后的章节中还要详细介绍。

2.1　控 制 流 图

计算机程序设计语言一般都提供一些基本的控制结构用以表达程序中的控制流。例如，顺序结构、条件结构和循环结构就是其中最典型的代表。在顺序结构中，每条语句都应该按照它们在程序中的先后顺序被执行；在条件结构中，if 语句（或 case 语句）的表达式确定哪一个（或哪些）分支将被执行；在循环结构中，do 语句、while 语句和 for 语句的表达式通常确定语句执行的条件及循环的次数等。为了准确地刻画程序中的控制流，以便准确地进行控制流和数据流分析，从而达到程序切片的目的，本小节重点介绍控制流图的基本概念和一些基本属性。基本模块是构造控制流图的基础，故先介绍基本模块的概念。

2.1.1　基本模块

在程序中，通常有一些连续的多行代码，它们形成程序中一个个相对独立的构造（constructs），并且具有特性：只能通过构造的第一条语句进入，通过最后一条语句离开该构造。所以有：

定义 2.1　一个基本模块就是一组连贯的语句，其中的控制流在模块开始时进入，在模块结束时离开，而没有在除模块结束之外的地方停止或出现分支的可能性。

定义中“一组连贯的语句”是指程序中一列“连续顺序执行的语句”。例如，在图 2.1 所示的程序片段中，前两条语句就构成一个程序的基本模块。

```
g: =f
h: =t-g
if (e>0) then
    h=t+g
```

图 2.1　代码片段

每个基本模块有且只有一个入口和出口，控制流只能从模块的入口进入基本