第35卷 第21期 计 算 机 工 程 2009年11月
Vol.35 No.21 Computer Engineering November 2009
·博士论文· 文章编号：1000—3428(2009)21—0013—04 文献标识码：A 中图分类号：N945.12

过程蓝图程序表示模型与视图导出方法

刘建宾
(北京信息科技大学计算机学院软件工程系，北京 100101)

摘 要：针对传统程序技术长期存在的表示分离、开发低效、质量欠佳、维护困难等问题，提出一种跨越分析、设计和构造阶段的多阶段程序过程表示模型。采用概念、逻辑和实现3层抽象视图、控制流和数据流二级映射的架构及视图导出方法，通过建立映射约束规则和对应关系，实现视图间的联系与统一，保证导出制品的一致性和有效性，使增量迭代、逐步求精的模型驱动设计过程规范化，有效提高程序生产率与可维护性。

关键词：过程蓝图；多阶段程序表示模型；建模语言；模型驱动程序设计；视图导出

Program Representation Model and View Derivation Approach for Procedure Blueprint

LIU Jian-bin
(Dept. of Software Engineering, Computer School, Beijing Information Science and Technology University, Beijing 100101)

【Abstract】 Aiming at common problems such as separate representation, poor efficiency and quality, difficult maintenance existed in traditional programming technologies for a long time, this paper proposes a kind of multi-phrase program procedure representation model across the analysis, design and implementation phrases with the architecture of three abstract views at conceptual, logical and implementing level, and two-level mappings on the control flow and data flow, and its views derivation approaches for the procedure blueprint. By means of establishing mapping constraint rules and corresponding relationship among the views, the procedure blueprint not only realizes association and unification of the outer views, but also assures validity and consistency of the derived artifacts, normalizing incremental, iterative and step by step refinement model-driven programming process, and improving productivity and maintainability of program effectively.

【Key words】 procedure blueprint; multi-phrase program representation model; modeling language; model driven programming; view derivation

1 概述

在实际的软件开发中，不仅要全面宏观地分析系统中的各种对象类及其关系，而且还要为每个对象类的所有操作设计出具体实现算法的过程。特别是在算法密集的复杂程序开发中，用得最多的还是过程的设计。为过程的设计与开发提供更有效的表现技术与手段，建立可视化过程建模语言与设计方法，始终是软件工程学研究的重要主题和方向。迄今为止，尽管已出现为数众多的过程描述语言和方法[1-2]，但能够同时跨越分析、设计和构造开发阶段的多阶段统一描述方法却非常罕见。在开发的不同阶段，用户需要在许多相似的过程描述语言中进行选择，其结果是在不同的开发阶段分别使用不同的描述语言进行表示。由于不同表示法彼此间缺乏联系，且在结构、形式和抽象程度等方面存在差异，导致不同表示法之间的过渡转换和一致性维护变得较为困难，容易出错且代价高昂；同时，后续阶段需重写前一个阶段已定义的结构和语义，造成工作的重复浪费并降低开发效率。

经过多年研究，笔者在结构化语言、软件蓝图设计语言、活动图、PAD图、PDL(伪码)、UML动作语义[3]以及多种现代文本编程语言的基础上发展出一种新的多阶段统一过程描述语言——过程蓝图建模语言[4]。

2 程序过程的表示

2.1 控制流与数据流

程序的控制流和数据流是构成过程的2个基本构件。控制流是程序为完成处理或计算所执行的数据操作的次序，即数据操作的序列。数据流表示数据元素中的值怎样被求值、改变和移动，即数据元素值的序列。它是执行数据类型操作、数据存储操作，以及数据流语句的结果。

2.2 抽象表现层次

为满足分析、设计和实现3个开发阶段对过程不同抽象和详细程度表示的需求，过程蓝图将程序过程的表示划分为概念、逻辑和实现3个层次，并使它们与3个开发阶段形成对应关系。概念层以自然且抽象的方式表现处理的概貌且不涉及具体细节，表达与编程实现语言无关的模型。逻辑层使用结构化控制构造描述算法过程的详细逻辑结构和工作流程，表达控制结构与编程实现语言相关的模型。实现层给出精确的实现细节，表达控制结构和数据流均与编程实现语言相关的模型。

2.3 表示方法

过程蓝图设计了3种外部视图——抽象概念结构图

基金项目：北京市自然科学基金资助项目(4073033)；北京市教委科技发展计划基金资助项目(KM200710772005)；北京市属市管高等学校人才强教计划基金资助项目；北京市属市管高等学校人才强教深化计划基金资助项目；广东省自然科学基金资助项目(05008310)
作者简介：刘建宾(1963－)，男，教授、博士，主研方向：程序建模语言，模型重构，程序理论与方法，软件工具与环境
收稿日期：2009-05-08 **E-mail：**ljb@bistu.edu.cn

(Abstract Concept Structure Diagram, ACSD)、抽象逻辑结构图(Abstract Logic Structure Diagram, ALSD)和抽象实现结构图(Abstract Implementation Structure Diagram, AISD)，作为 3 个抽象表现层次的过程表示工具。其中，ACSD 用于表示平台无关模型(Platform Independent Model, PIM)[5]，ALSD 和 AISD 用于表示平台相关模型(Platform Specific Model, PSM)[5]。

对控制流的表示，过程蓝图设计了概念和逻辑二级抽象的控制构造。ACSD 使用实现无关的概念控制构造描述控制流，而 ALSD 和 AISD 使用实现相关的逻辑控制构造描述控制流。

对数据流的表示，过程蓝图的 3 种视图分别采用自然语言、受限自然语言及目标编程语言作为数据流的描述形式，较自然地满足相应层次数据流表示的不同精度要求。

过程蓝图的抽象表现层次与表示方法可汇总为表 1。

表 1　过程蓝图的抽象表现层次与表示方法

层次	形式	内容	描述方式	特性	类别
概念层	ACSD	控制流	概念控制构造	实现无关	PIM
		数据流	自然语言	实现无关	
逻辑层	ALSD	控制流	逻辑控制构造	实现相关	PSM
		数据流	受限自然语言	实现无关	
实现层	AISD	控制流	逻辑控制构造	实现相关	PSM
		数据流	编程语言表达式	实现相关	

表 2 给出从 PASCAL, C, Java 等常用编程语言的控制语句中提取的二级控制构造类型及说明。

表 2　过程蓝图的控制构造类型

构造分类	概念类型	逻辑类型	说　明
顺序	+	SEQ	顺序结构
循环	*	WDO	While 循环结构
		FDO	For 循环结构
		RPT	Repeat 循环结构
可选	o	IFT	If then 结构
选择	?	IFE	If then else 结构
		CAS	多项选择结构
		WHN	选择情况分支结点
		OTW	否则情况分支结点
并发	&	PAR	并发处理结构
未定义	#	UND	未定义结点
块	!	CAL	模块调用结点
		SEG	过程段结点
跳转	>	ESC	退出结点
		NXT	继续循环结点
		RET	返回结点
终结点	:	OPE	操作结点
		DCL	数据声明/定义结点

3　程序表示模型

3.1　基本集合和函数

定义如下的集合和函数，为过程蓝图程序表示形式模型的定义建立基础。

(1)概念结点类型集合 T_{cn}={+, *, o, ?, !, :, >, &, #}。

(2)逻辑结点类型集合 T_{ln}={SEQ, WDO, FDO, RPT, IFT, IFE, CAS, WHN, OTW, CAL, SEG, OPE, DCL, ESC, NXT, RET, PAR, UND}。将 T_{ln} 划分成 2 个不相交的集合 $T_{ln}=T'_{ln}\cup T''_{ln}$, $T'_{ln}\cap T''_{ln}=\varnothing$。$T''_{ln}$={SEQ, OTW, ESC, NXT, PAR, SEG, UND}为不带有数据流的逻辑结点类型集合。$T'_{ln}=T_{ln}-T''_{ln}$ 为带有数据流的逻辑结点类型集合。T_{cn}, T_{ln} 各元素的含义见表 2。

(3)自然语言语句集合 S_{nn}={s｜s∈自然语言语句}。

(4)受限自然语言语句集合 S_{df}={s ｜ s∈S_{nn}∧verb(s)∈VDD∧noun(s)∈EDD∧conj(s)∈CDD}。其中，VDD, EDD, CDD 分别表示命令动词词典、程序实体名词典和连接词词典；verb: S_{df}→VDD; noun: S_{df}→EDD; conj: S_{df}→CDD。

(5)编程语言集合 PL={pl｜pl∈程序设计语言名字}。

(6)目标编程语言的基本操作命令和表达式集合 S_{pl}={s｜s∈目标语言表达式∨s∈目标语言基本操作语句}。其中，基本操作语句不包括结构化语句、GOTO 语句和标号语句。

(7)树函数：parent(t,n)返回树 t 的结点 n 的父结点，childnum(t,n)返回 n 结点的孩子结点数，brothernum(t,n)返回 n 结点的兄弟结点数，eldest(t,n)为判断结点 n 是否为最长兄弟结点的逻辑函数。最长兄弟结点是兄弟结点中结点序号最大的结点，unique(t,root)判断 root 结点是否是树 t 中唯一的没有父结点的结点，ancestor(t,n)返回结点 n 的祖先结点集合。

(8)分析函数：language_type(x)返回表达式 x 的语言类型，syntax_check(pl, exp)检查表达式 exp 是否符合 pl 语言的词法语法规则。

3.2　程序表示模型

定义 1 过程蓝图表示的程序 pb 是如下定义的六元组:

pb=(t_c=(A_c,dc), f_{cl},t_l=(A_l= $A_{l1}\cup A_{l2}$,dl), C_{df},f_{sc},t_i=(A_i= $A_{i1}\cup A_{i2}$,di))

其中，t_c, t_l, t_i 分别表示抽象概念结构图、抽象逻辑结构图和抽象实现结构图；A_c,A_l,A_i 分别表示概念结点、逻辑结点与实现结点的集合；f_{cl}: A_c→A_l 是概念结点到逻辑结点的一对一映射函数，并且它满足下面定义的映射约束规则 R_{cl}；C_{df} 是用目标编程语言表示的操作表达式表，$C_{df}\subset S_{pl}$；f_{sc}: A_{l1}→C_{df} 是带数据流的逻辑结点到操作表达式表的映射函数。下面给出 t_c, t_l, t_i 及 R_{cl} 的定义。

(1)抽象概念结构图

定义 2 抽象概念结构图是用 t_c=(A_c,dc)表示的、由概念结点构成的树，是过程蓝图 pb 的概念视图。其中，$A_c\subset T_{cn}\times S_{nn}$ 是概念结点的集合；dc: A_c→B_c 是满足树条件和下面定义的概念结点分解条件的概念结点分解函数。B_c={dc(a)| ∀a∈A_c}为 A_c 中所有结点的子结点系列 dc(a)的集合。dc(a)={a_1,a_2,…, a_k}=Sequence{a_j｜j≥0∧j≤k∧a_j∈ A_c 为 a 结点的子结点，k≥0 为结点 a 的子结点数}。

对 a∈A_c，dc(a)应满足如下的条件：

1)若 a[T_{cn}]∈{:,>,!,#}，则|dc(a)|=0，称 a 为叶概念结点；若 a[T_{cn}]∈{+, *, o, ?, &}，称 a 为复合概念结点，且若|dc(a)|=0，则称 a 为未分解复合概念结点；否则称 a 为已分解复合概念结点。对已分解复合概念结点，必须满足下面的条件。

2)若 a[T_{cn}]="?"，则｜dc(a)｜≥2，且当｜dc(a)｜=2 时，对 a_j ∈dc(a)，a_j[T_{cn}]∈T_{cn}(j=1,2)；当｜dc(a)｜>2 时，对 a_j∈dc(a)，有 a_j [T_{cn}]∈{+,o,#}(j=1,2,…,k)。

3)若 a[T_{cn}]∈{+, *, o}，则｜dc(a)｜≥1，且对 a_j∈dc(a)，有 a_j [T_{cn}]∈T_{cn} (j=1,2,…,k)。

4)若 a[T_{cn}]="&"，则｜dc(a)｜≥2，对 a_j∈dc(a)，a_j[T_{cn}]∈ T_{cn} (j=1,2,…,k)。

(2)抽象逻辑结构图

定义 3 抽象逻辑结构图是用 t_l=(A_l= $A_{l1}\cup A_{l2}$,dl)表示的、由逻辑结点构成的树，是过程蓝图 pb 的逻辑视图。其中，$A_{l1}\subset T'_{ln}\times S_{df}$ 是带有数据流的逻辑结点集合；$A_{l2}\subset T''_{ln}\times S_{nn}$ 是不带有数据流的逻辑结点集合；dl: A_l→B_l 是满足树条件和如下定义的逻辑结点分解条件的逻辑结点分解函数。B_l ={dl(a)| ∀a∈A_l}为 A_l 中所有结点的子结点系列 dl(a)的集合。dl(a)={a_1,a_2,…,a_k}=Sequence{a_j｜j≥0∧j≤k∧a_j∈A_l 为 a 结点的子结点，k≥0 为结点 a 的子结点数}。

对 $a\in A_l$，$dl(a)$应满足下列条件：

1)若 $a[T_{ln}]\in\{OPE, DCL, CAL, SEG, ESC, NXT, RET, UND\}$，则有$|dl(a)|=0$，称 a 为叶逻辑结点。若 $a[T_{ln}]\in\{SEQ, WDO, FDO, RPT, IFE, IFT, CAS, WHN, OTW, PAR\}$，称 a 为复合逻辑结点，且当$|dl(a)|=0$ 时，称 a 为未分解复合逻辑结点，当$|dl(a)|\neq 0$ 时，称 a 为已分解复合逻辑结点。对已分解复合逻辑结点还有下列条件。

2)若 $a[T_{ln}]$="IFE"且$|dl(a)|\neq 0$，则有$|dl(a)|=2$，且对 $a_j\in dl(a)$，有 $a_j[T_{ln}]\in T_{ln}$-{WHN, OTW},j=1,2。

3)若 $a[T_{ln}]\in\{SEQ, WDO, FDO, RPT, IFT, WHN, OTW\}$且$|dl(a)|\neq 0$，则有$|dl(a)|\geq 1$，且对 $a_j\in dl(a)$，有 $a_j[T_{ln}]\in T_{ln}$-{WHN, OTW},j=1,2,…,k。

4)若 $a[T_{ln}]$="PAR"且$|dl(a)|\neq 0$，则有$|dl(a)|\geq 2$，且对 $a_j\in dl(a)$，有 $a_j[T_{ln}]\in T_{ln}$-{WHN, OTW},j=1,2,…,k。

5)若 $a[T_{ln}]$="CAS"且$|dl(a)|\neq 0$，则有$|dl(a)|\geq 2$，且对 $a_j\in dl(a)$,j=1,2,…,k，有 $a_j[T_{ln}]$="WHN",j=1,2,…,k-1, $a_k[T_{ln}]\in$ {WHN, OTW}。

(3)操作表达式表 C_{df} 和映射函数 f_{sc}

操作表达式表 C_{df} 给出用目标编程语言表示的算法过程数据流的具体实现细节，包括实现形式的数据流表达式和基本操作表达式。映射函数 f_{sc} 则在抽象逻辑结构图 t_l 与目标编程语言数据流表达式的实现细节之间建立联系。

(4)概念结点到逻辑结点的映射约束规则 R_{cl}

$R_{cl}=\{R(x)\mid x\in T_{cn}\}$

对 $a\in A_c$，$f_{cl}(a)$应满足约束规则集 $R(a[T_{cn}])$。

1)顺序结点映射约束规则集 R(+)={r1,r2,r3}

r1:parent(t_c,a)[T_{cn}]="?" ∧ brothernum(t_c,a) ≥ 3 ∧ not eldest(t_c,a) -> $f_{cl}(a)[T_{ln}]$="WHN"

r2:parent(t_c,a)[T_{cn}]="?" ∧ brothernum(t_c,a) ≥ 3 ∧ eldest (t_c,a) -> f_{cl} (a)[T_{ln}]="WHN"∨f_{cl} (a)[T_{ln}]="OTW"

r3:其他情况 $f_{cl}(a)[T_{ln}]$="SEQ"

2)循环结点映射约束规则集 R(*)={r4}

r4:$f_{cl}(a)[T_{ln}]$="WDO"∨$f_{cl}(a)[T_{ln}]$="FDO"∨$f_{cl}(a)[T_{ln}]$= "RPT"

3)选择结点映射约束规则集 R(?)={r5,r6}

r5:childnum(t_c,a)=2 ∧ ∃cn∈dc(a)(cn[T_{cn}] ≠ "o") -> $f_{cl}(a)[T_{ln}]$= "IFE"

r6:其他情况 $f_{cl}(a)[T_{ln}]$="CAS"

4)可选结点映射约束规则集 R(o)={r7,r8,r9}

r7:parent(t_c,a)[T_{cn}]="?"∧not eldest(t_c,a)∧(brothernum(t_c, a)>2 ∨ brothernum(t_c,a)=2 ∧ ∀bn∈dc(parent(t_c,a))(bn[T_{cn}]= "o")) -> $f_{cl}(a)[T_{ln}]$="WHN"

r8:parent(t_c,a)[T_{cn}]="?" ∧ eldest(t_c,a) ∧ (brothernum(t_c,a)> 2∨brothernum(t_c,a)=2∧∀bn∈d c(parent(t_c,a))(bn[T_{cn}]="o"))-> $f_{cl}(a)[T_{ln}]$="WHN"∨$f_{cl}(a)[T_{ln}]$="OTW"

r9:其他情况 $f_{cl}(a)[T_{ln}]$="IFT"

5)跳转结点映射约束规则集 R(>)={r10}

r10:$f_{cl}(a)[T_{ln}]$="ESC"∨$f_{cl}(a)[T_{ln}]$="NXT"∨$f_{cl}(a)[T_{ln}]$= "RET"

6)块结点映射约束规则集 R(!)={r11}

r11:$f_{cl}(a)[T_{ln}]$="CAL"∨$f_{cl}(a)[T_{ln}]$="SEG"

7)终结点映射约束规则集 R(:)={r12}

r12:$f_{cl}(a)[T_{ln}]$="OPE"∨$f_{cl}(a)[T_{ln}]$="DCL"

8)并发结点映射约束规则集 R(&)={r13}

r13:$f_{cl}(a)[T_{ln}]$="PAR"

9)未定义结点映射约束规则集 R(#)={r14,r15,r16}

r14:parent(t_c,a)[T_{cn}]="?" ∧ brothernum(t_c,a)>2 ∧ not eldest(t_c,a) -> $f_{cl}(a)[T_{ln}]$="WHN"

r15:parent(t_c,a)[T_{cn}]="?" ∧ brothernum(t_c,a)>2 ∧ eldest(t_c, a) -> $f_{cl}(a)[T_{ln}]$="WHN"∨$f_{cl}(a)[T_{ln}]$="OTW"

r16:其他情况 $f_{cl}(a)[T_{ln}]\in T_{ln}$-{WHN,OTW}

(5)抽象实现结构图

定义 4 抽象实现结构图是用 $t_i=(A_i=A_{i1}\cup A_{i2},di)$表示的、由实现结点构成的树，是过程蓝图 pb 的实现视图。其中，$A_{i1}\subset T'_{ln}\times S_{pl}$ 是带数据流的实现结点集合；$A_{i2}\subset T''_{ln}\times S_{nn}$ 是不带数据流的实现结点集合，$A_i=A_{i1}\cup A_{i2}$ 是满足实现结点表示规则的实现结点集合；di: $A_i\rightarrow B_i$ 且满足树条件和实现结点分解条件的实现结点分解函数。$B_i=\{di(a)\mid \forall a\in A_i\}$为 A_i 中所有结点的子结点系列 di(a) 的集合。$di(a)=\{a_1,a_2,\cdots,a_k\}$= Sequence$\{a_j\mid j\geq 0\wedge j\leq k\wedge a_j\in A_i$ 为 a 结点的子结点，$k\geq 0$ 为结点 a 的子结点数$\}$。

1)实现结点表示规则

①使用的逻辑控制构造类型和数据流表达式属于同一种编程语言。

$\exists X\in PL(\forall a\in A_i((language\_type(a[T_{ln}])=X) \wedge \forall a\in A_{i2}(language\_type(a[S_{pl}])=X)))$

②书写的数据流操作表达式符合编程语言的词法、句法和表达式语法规则。

$\forall a\in A_{i2}(syntax\_check(language\_type(a[T_{ln}]), a[S_{pl}])= true)$

2)实现结点分解条件

与逻辑结点分解条件相似，不再列出。

4 视图导出方法

基于过程蓝图的程序设计过程是从概念层模型出发逐步细化并导出逻辑层模型和实现层模型的模型驱动过程。设计者首先使用实现无关的抽象概念结构图描述程序的抽象算法，得到程序的概念模型 t_c；然后从 t_c 出发导出控制结构与实现相关的初始抽象逻辑结构图，并进一步对未定义和未分解复合逻辑控制结构细化展开，得到结点类型完全确定并彻底分解的抽象逻辑结构图 t_l；在此基础上，通过构造操作表达式表 C_{df} 和映射函数 f_{sc} 对数据流求精并建立 t_l 与 C_{df} 之间的联系，最终导出控制流和数据流两方面均与实现相关的抽象实现结构图 t_i，完成程序设计工作。下面给出从高层模型 t_c 出发，逐步细化并导出 t_l 和 t_i 这 2 种较低层模型视图的基本方法。

4.1 初始抽象逻辑结构图的导出

可从 t_c 出发导出一个与 t_c 结构一致的初始 t_l。

(1)初始 A_l 和 f_{cl} 的构造

对 t_c 中的每个概念结点 a, $a\in A_c$，首先需要运用概念结点到逻辑结点的映射规则 R_{cl} 中的规则集 $R(a[T_{cn}])$将结点 a 映射成逻辑结点 a'，使 $a'\in A_l$，然后在 a 与 a'之间建立对应关系，使 $f_{cl}(a)=a'$。当 t_c 中的所有结点进行了映射时，就完成了初始 A_l 和 f_{cl} 的构造工作，而且可以根据 A_l 构造出 A_{l1} 和 A_{l2}。$A_l=\{f_{cl}(a)\mid \exists a(a\in A_c)\}$，$A_{l1}=\{a\mid \exists a(a\in A_l \wedge a[T_{ln}]\in T'_{ln})\}$，$A_{l2}=\{a\mid \exists a\ (a\in A_l\wedge a[T_{ln}]\in T''_{ln})\}$。

(2)dl 的初始构造

对 t_c 中的每个概念结点 $a\in A_c$，若 $dc(a)=\varnothing$，则 $dl(f_{cl}(a))=\varnothing$；否则设 $dc(a)=\{a_j\mid j=1,2,\cdots,k\}$，让 $dl(f_{cl}(a))=\{f_{cl}(a_j)\mid j=1,2,\cdots,k\}$。

按上述构造方法得到的 t_l 称为导出 t_l。A_l 和 dl 的构造方

法保证了导出 t_l 与 t_c 同构且相关结点类型满足映射约束规则 R_{cl}。对定义中的有关条件、规则的无矛盾性，也已证明。

定理 1 当 t_c 满足树条件和概念结点分解条件时，满足映射约束规则 R_{cl} 的导出 t_l 一定满足树条件和逻辑结点分解条件。

4.2 初始抽象逻辑结构图的细化与确定化

通过将导出 t_l 中的未定义结点确定为具体类型，以及对未分解复合逻辑结点展开子结构的活动完成对抽象逻辑结构图的细化，并要求每一步细化时，保持 t_l 树条件和逻辑结点分解条件。当 t_l 中不存在任何未定义结点和未分解复合逻辑结点时，细化工作结束并得到确定的 t_{ls}。确定的抽象逻辑结构图是所有结点类型完全明确且复合结构已彻底分解的树。

4.3 操作表达式表 C_{df} 和映射函数 f_{sc} 的构造

对 C_{df} 和 f_{sc}，可从抽象逻辑结构图 t_l 出发构造出来。构造方法是：对抽象逻辑结构图 t_l 上的每个带数据流的逻辑结点，即 A_{l1} 中的每个结点 $a(a\in A_{l1})$，进行如下处理：首先构造一个操作表达式表的新表项 c，然后用目标程序语言写出结点 a 的数据流表现形式(表达式或基本操作)并填入 c 中，最后在结点 a 和表项 c 间建立联系，使 $f_{sc}(a)=c$。当 A_{l1} 中的所有结点都构造完毕时，可得到与抽象逻辑结构图 t_l 相对应的操作表达式表 C_{df}，以及 $A_{l1}\rightarrow C_{df}$ 的映射函数 f_{sc}，且 $C_{df}=\{f_{sc}(a)\mid \exists a(a\in A_{l1})\}$。

4.4 抽象实现结构图的导出

t_i 是 t_l,C_{df} 和 f_{sc} 的构造结果和集成视图。

(1)A_i 的构造：为了方便 di 的构造，在构造 A_i 的同时，构造一个 $A_l\rightarrow A_i$ 的 1:1 函数 f_{li}。对 t_l 中的每个逻辑结点 $a\in A_l$，若 $a\in A_{l2}$，则产生一个新结点 $a'\in A_{i2}$，使 $a'=a$, $f_{li}(a)=a'$；否则产生 A_{i1} 的新结点 a'，使 $a'[T_{ln}]=a[T_{ln}]$, $a'[S_{pl}]=f_{sc}(a)$, $f_{li}(a)=a'$。

(2)di 的构造：对 t_l 中的每个逻辑结点 $a\in A_l$，若 $dl(a)=\varnothing$，则 $di(f_{li}(a))=\varnothing$；否则设 $dl(a)=\{a_j \mid j=1,2,\cdots,k\}$，让 $di(f_{li}(a))=\{f_{li}(a_j)\mid j=1,2,\cdots,k\}$。

从 A_i 和 di 的构造方法可知 t_i 与 t_l 的结构一致，结点类型相同。从 t_i 与 t_l 的定义可看到，t_i 的实现结点分解条件与 t_l 的逻辑结点分解条件同质，因而可得结论如下：

定理 2 当 t_l 满足树条件和逻辑结点分解条件，定有从 t_l, C_{df} 和 f_{sc} 的导出的 t_i 满足树条件和实现结点分解条件。

5 结束语

本文在论述控制流和数据流过程构件基本概念、过程蓝图的抽象层次和表示方法的基础上，给出过程蓝图的程序表示模型以及视图的导出方法。过程蓝图将程序过程的表示分为概念、逻辑和实现 3 个抽象层次，并分别使用抽象概念结构图、抽象逻辑结构图和抽象实现结构图作为外部表示视图，同时通过三层视图间的控制流和数据流的二级映射，将 3 个外部视图联系成为 1 个统一整体并保证导出视图的有效和一致。统一的程序表示体系结构和视图导出方法，一方面能满足不同开发阶段对过程表示的不同精度和一致性要求，避免结构重写和一致性维护的高昂代价，并使增量迭代逐步求精的程序设计过程在求精层次和内容方面更加规范化，十分利于组织工程化开发；另一方面，不仅为模型驱动开发语境下的 PIM, PSM 模型，以及模型重构语境下的分层抽象重构操作提供程序过程模型表达的语言基础，还为开发阶段的平滑过渡、模型转换和代码生成提供可行的解决方案和实现机制。

过程蓝图目前已在一些中小规模的程序开发和大学程序设计相关课程的教学中得到初步应用[4]，实践表明它能提高 10%~30%左右的程序开发效率，减少 50%左右的设计逻辑错误，并使程序的可维护性得到显著提高。

参考文献

[1] Wieringa R. A Survey of Structured and Object-oriented Software Specification Methods and Techniques[J]. ACM Computing Surveys, 1998, 30(4): 459-527.

[2] Tripp L L. A Survey of Graphical Notations for Program Design——An Update[J]. ACM Software Engineering Notes, 1988, 13(4): 110-115.

[3] Object Management Group. Action Semantics for UML[EB/OL]. (2002-01-09). www.omg.org/docs/ptc/02-01-09.pdf.

[4] 刘建宾. 过程蓝图设计方法及其支撑工具[D]. 西安: 西北大学, 2004.

[5] Kleppe A, Warmer J, Bast W. MDA Explained: The Practice and Promise of the Model Driven Architecture[M]. Boston, USA: Addison Wesley, 2003.

编辑　任吉慧

（上接第 12 页）

图 6 显示当初始值改变的情况下线性复杂度基本保持稳定，图 7 是图 6 的局部放大图，可以看出在 x=0.796 348 和 x=0.796 348 000 000 09 的条件下，生成序列的线性复杂度又有很大的不同，说明序列非常敏感。

这种序列所具有的敏感性从相关攻击的角度来说很难确定其产生方法，即使碰巧使用了准确的产生被攻击混沌序列的混沌映射，但是若其初始值不能精确得到，只要两者有微小的误差，由混沌对初值和参数的极端敏感依赖性，所得到的混沌攻击序列与被攻击混沌序列的互相关也很小，从而无法进行相关攻击和干扰所预期的混沌序列。

4 结束语

用 Legendre 序列来对混沌序列进行扰动的研究还很少，本文对这种想法进行初步的分析与实现。统计结果与理论分析表明，本方法产生的混沌序列有良好的线性特性，且软件实现简单。传统的线性移位寄存器产生的序列周期是限定的，而且数量较少，而混沌序列的周期是任意的，且数量极大。在线性复杂度上，传统序列的复杂度为移位寄存器的级数，而混沌序列的线性复杂度为所需序列长度的一半左右，更加灵活和难以预测。同时，大素数、Legendre 扰动的引入改善了混沌序列短周期现象，也改善了由于计算机精度问题所带来的缺陷，为混沌映射在密码加密中的应用提供了新的思路。

参考文献

[1] 斯皮尔曼. 经典密码学与现代密码学[M]. 曹　英, 叶阮健, 张长富, 译. 北京: 清华大学出版社, 2005: 97-101.

[2] 周　悦. M 序列置乱对数字混沌序列性能的影响[J]. 信息技术, 2007, 3(8): 81-83.

[3] 柳　平, 闫　川, 黄显高. 改进的基于 Logistic 映射混沌扩频序列的产生方法[J]. 通信学报, 2007, 28(2): 134-140.

[4] 王育民, 何大可. 保密学基础与应用[J]. 西安: 西安电子科技大学出版社, 1990.

[5] 杨义先, 林须端. 编码密码学[M]. 北京: 人民邮电出版社, 1992.

[6] Lipton J M, Dabke K P. Spread Spectrum Communications Based on Chaotic Systems[J]. International Journal of Bifurcation and Chaos, 1996, 6(12): 361-374.

编辑　任吉慧