第32卷 第15期 计 算 机 工 程 2006年8月
Vol.32 № 15 Computer Engineering August 2006
·软件技术与数据库· 文章编号：1000—3428(2006)15—0081—03 文献标识码：A 中图分类号：TP311.53

基于依赖性分析的UML状态图切片技术

卢炎生，王 曦，谢晓东，毛澄映
(华中科技大学计算机科学与技术学院，武汉 430074)

摘 要：将 UML 状态图、程序切片和软件测试技术三者结合起来，生成基于依赖性分析的 UML 状态图切片，为软件测试中待测试的程序的理解、发现和修改错误提供了一种方法。最后运用实例讨论了该切片方法的实用性，应用于回归测试中具有较高的效率。
关键词：统一建模语言；UML 状态图；切片；依赖性分析

UML Statechart Diagram Slicing Based on Dependency Analysis

LU Yansheng, WANG Xi, XIE Xiaodong, MAO Chengying
(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074)

【Abstract】 This paper combines the three techniques UML statechart diagram, program slicing and software testing together to generate the UML statechart slices that based on the dependency analysis. It provides a method for the program comprehension, errors finding and modification during the process of sofrware testing. At last the case study is given to show the applicability and higher effiency of the slicing method used in the regression test.
【Key words】 UML; UML statechart diagram; Slice; Dependency analysis

传统的软件技术主要是基于数据流分析和控制流分析，这类测试技术很难保证测试的充分性和完备性，并且程序的微小变动都将引起大量的测试工作。近年来，基于“程序切片”的软件测试研究逐渐引起人们的注意。“程序切片”的概念最早是由 Mark Weiser提出的。作为一种重要的程序分析理解方法[2]，程序切片的概念主要应用在程序调试和理解方面，而在程序的测试中的应用还不多见，其中作出较大贡献的有B.Korel, M. Kamkar,J.R. Lyle和M.J.Harrold等。B.Korel[3]通过依赖性分析，提出了基于模型的减少回归测试用例的方法。Gupta,Harrold 和Soffa[4]阐述了利用切片技术进行回归测试的方法：通过在改动的地方计算前向切片和后向切片判别受改动影响的程序——只有那些执行了受影响的定义引用对的测试用例需要重新执行，这就大大减少了测试工作。

UML状态图刻画了系统中对象在其生命周期中的行为和状态变迁，在软件分析和设计建模中占有重要地位[1]。UML模型可以作为测试阶段的依据，不同的测试可以使用不同的图作为测试依据。

从程序切片和 UML 状态图着手，根据测试人员的需要，通过对面向对象程序进行切片来分解程序，把对整个程序的测试按照一定的规则转化为只对 UML 状态图切片的测试。可以说明的是，这种以切片为基础的测试技术比单纯利用数据流分析和控制流分析的传统测试技术要优越得多，尤其是在回归测试中。

1 基于依赖性分析的 UML 状态图的切片算法

1.1 依赖性分析

在提出 UML 状态切片的计算方法前，对 uml 状态图存在的依赖进行依赖性分析。我们在变迁 f(f 是成员方法)中定义两种依赖性：数据依赖和控制依赖。在定义依赖性以前，先给出几个相关定义。

G(P)是程序 P 的集合，V 是一组程序变量。

定义 1 定义节点。节点n∈G(P)是变量v∈V的定义节点，记作DEF(f_i,v,n)，当且仅当变量v的值由对应节点n的语句片段处定义，f为对应的成员方法。定义节点的集合记为 S_{di}，即：

$S_{di}=\{DEF(f_i,v,n)\mid f$ 为对应的成员方法，$v\in V$，$n\in G(P)$，且 v 的值由其对应节点 n 的语句片断处定义$\}$。

定义 2 使用节点。节点n∈G(P)是变量v∈V的使用节点，记作USE(f_i,v,n)，当且仅当变量v的值在对应节点n的语句片段处使用，f为对应的成员方法。使用节点的集合记为S_{ui}，即：

$S_{ui}=\{USE(f_i,v,n)\mid f_i$为对应的成员方法，$v\in V$，$n\in G(P)$，且 v 的值由其对应节点 n 的语句片断处定义$\}$。

定义 3 f_i 为UML状态图中的变迁，我们对UML状态图上的变迁f_i的进行扩展定义，即表示为$f_i(S_{di},S_{ui})$,其中S_{di} 为定义节点集合，S_{ui}为使用节点集合。

存在数据依赖指的是在一个变迁中对某个变量的值进行定义，而在另一变迁中用到这个变量的值。

设Path(P)是UML状态图中f_i到f_j的路径集合，即Path(P)＝{p_i | p_i 是UML状态图中f_i到f_j的路径}。对变迁f_i与f_j存在数

基金项目：“十五”国家级科技预研基金资助项目(41315.9.2)；湖北省自然科学基金资助项目
作者简介：卢炎生(1949—)，男，教授、博导，主研方向：现代数据库系统，软件测试与构件技术，数据挖掘；王 曦，硕士生；谢晓东，讲师、博士；毛澄映，博士生
收稿日期：2005-11-01 **E-mail：**wangxi123@sohu.com

据依赖作如下定义：

定义 4 变迁f_i与f_j存在数据依赖,当且仅当：

(1)$v \in S_{di}$；

(2) $v \in S_{uj}$；

(3)$\exists p_i \in Path(P)$，使得v的值保持不变。

控制依赖原先是在基于程序控制流图定义而来的。在控制流图中的一个结点i可以影响到另一个结点j的执行，则称j控制依赖于i。本文对UML状态图中的控制依赖定义进行扩展。在UML状态图中的控制依赖存在于变迁中，即一个变迁f_i可能影响到另一个变迁f_j的触发。在变迁之间的控制依赖与程序控制流图中结点之间的控制依赖类似。

$P_s=\{c_i \mid c_i$是状态S_i到终止状态的一条路径$\}$，用$\rightarrow_{control}$表示“ 控制依赖于 ”，用 $\not\rightarrow_{control}$表示“ 不控制依赖于 ”。

定义 5 状态的控制依赖[6]。设S_i与S_j为两个状态，f_i为从S_i发出的变迁：

(1)$S_j \rightarrow_{control} S_i$ iff $\forall c_i \in P_s, S_j \in c_i$；

(2)$S_j \rightarrow_{control} f_i$ iff $\forall c_i \in P_s, f_i \in c_i, S_j \in c_i$。

定义 6 变迁f_i与f_j之间的控制依赖，即f_j控制依赖于f_i，当且仅当：

(1)$S_b(f_i) \not\rightarrow_{control} S_b(f_j)$；

(2)$S_b(f_j) \rightarrow_{control} f_i$ 。

其中$S_b(f_i)$为触发变迁f_i的一个状态，$S_e(f_i)$为被变迁f_i触发的一个状态。

1.2 切片算法

1.2.1 切片标准

一个类的完整的 UML 状态图 SD 和给定的变迁 f(类中的方法 f)中的变量 v 构成了该类的 UML 状态图的切片准则<SD,f,v>。

对于某个类的 UML 状态图 SD，给定切片标准<SD,f,v>,如果一种切片 Slice 满足下列条件则称它为该类的状态图切片 SD’：

(1)Slice 中包含了该类的对象的状态和状态间的迁移；

(2)Slice 是一个具有一定语义的面向对象的状态图。

1.2.2 切片算法

给定 UML 的扩展状态图的模型 SD 和在变迁中的变量 v，目标是要自动产生一个关于变迁 f 中的变量 v 的 UML 状态图切片 SD’。SD’是符合切片标准 SD 的子图，它包括所有影响变迁 f 中变量 v 的所有变迁。这样，得到的 UML 状态图切片是一个可执行的模型。

步骤：

(1)切片 SD’是整个 UML 状态图模型 SD，并且所有的变迁在 SD’中标记为 not-involved；

(2)删去那些无法到达变迁 f 的变迁，显然，这些变迁是不会影响到变迁 f 的；

(3)通过控制依赖分析，找到所有与 f 有控制依赖关系的变迁，设存在控制依赖的变迁集合为 CD，即：

CD＝{变迁f_i与f_j存在控制依赖| $S_b(f_i) \not\rightarrow_{control} S_b(f_j), S_b(f_j) \rightarrow_{control} f_i$ }。

将在变迁集合 CD 内的变迁标记为 involved，显然这些变迁是会影响到 f 中的变量 v 的；

(4)通过数据依赖分析，找到所有与 f 中的变量 v 有数据依赖关系的变迁，设存在数据依赖的变迁集合为 DD,即：

DD＝{变迁f_i与f_j存在数据依赖| $v \in S_{di}$, $v \in S_{uj}$, $\exists p_i \in Path(P)$,使得v的值保持不变}。

将在变迁集合 DD 内的变迁标记为 involved。

算法描述：

```
Input：UML 状态图 exSD，变迁 f 中的变量 v
Output：UML 状态图 SD’
Begin
        SD＝SD’;
将 UML 状态图 SD’中所有的变迁标记为 not-involved;
    Do
    删去无法到达 f 的变迁；
While(找到无法到达变迁 f 的变迁);
End do

While ( 与变迁 f 有控制依赖关系)
    将其变迁标记为 involved；
End While
While(与变迁 f 有数据依赖关系)
    将其变迁标记为 involved；
End While
    显示 UML 状态图的切片 SD’；
End
```

2 实例分析

为了验证基于依赖性分析的 UML 状态图切片方法的可用性、实用性和优越性，在此用一个具体的案例来讨论该切片方法。不失一般性，采用文献[7]中的用 c++实现的关于售货机的例程。这个售货机它只收硬币，当前的硬币数为两个时可以进行售货。该售货机的 c++代码如下：

```
1 class CCoinBox{
2 unsigned totalQtrs,curQtrs,unsigned allowVend;
                    public:
3 CCoinBox(){Reset();}
4 void AddQtr();     //add a quarter
5     void ReturnQtrs() { curQtr=0;     allowVend=0;}
6     void Reset() { totalQtrs=allowVend=0;
7                     curQtrs=0;}
8     void Vend();
                    };
9     void CCoinBox::AddQtr() {
10    curQtrs=curQtrs+1;       //add a quarter
11    f(curQtrs>1)    allowVend=1;
                  }
12    void CCoinBox::Vend(){
13     if(allowVend)       //if allowVend
          { //update totalQtrs,curQtrs,allowVend
14    totalQtrs=totalQtrs+curQtrs;
15    curQtrs=allowVend=0;
                    }     //else no action
              }
```

下面对自动售货机的 C++源代码进行控制依赖分析和数据依赖性分析：

变量集合 V＝{ totalQtrs, curQtrs, allowVend}

选取状态变量：curQtrs,allowVend 。

以 AddQtr()为例，通过控制依赖分析找出与 AddQtr()存在控制依赖关系的变迁有 AddQtr()，ReturnQtrs()，Vend()。

由数据依赖性分析中的定义 1、定义 2 可得到 CCoinBox

类中的使用节点集合和定义节点集合：

对于 curQtrs 变量：

定义节点有：

DEF(ReturnQtrs,curQtrs,5),

DEF(Reset,curQtrs,7) ,

DEF (Vend,curQtrs,15)；

使用节点有：

USE(AddQtr,CurQtrs,10), USE(Vend,curQtr,11),

USE(Vend,curQtr,14)。

对于 allowVend 变量：

定义节点有：

DEF(ReturnQtrs,allowVend,5) ,

DEF(Reset,allowVend,6) ,

DEF(AddQtr,allowVend,11) ,

DEF(Vend,allowVend，15)；

使用节点有：

USE(Vend，allowVend,13)。

由上得使用节点集合为：

Sd={ DEF(ReturnQtrs,curQtrs,5),

DEF(Reset,curQtrs,7) ,

DEF (Vend,curQtrs,15)，

DEF(Reset,allowVend,6) ,

DEF(AddQtr,allowVend,11) ,

DEF(Vend,allowVend，15),

DEF(ReturnQtrs,allowVend,5) }

定义节点集合为：

Su={ USE(AddQtr,CurQtrs,10) ,

USE(Vend,curQtr,11) ,

USE(Vend,curQtr,14)，

USE(Vend，allowVend,13)}

根据定义 3，对 UML 状态图上的变迁(成员函数)进行扩展：

成员函数 AddQtr(Sd3,Su3)

Sd3={ DEF(AddQtr,allowVend,11) }

Su3={ USE(AddQtr,CurQtrs,10) }

其他成员函数同理。

得到扩展后的 UML 状态图，见图 1。

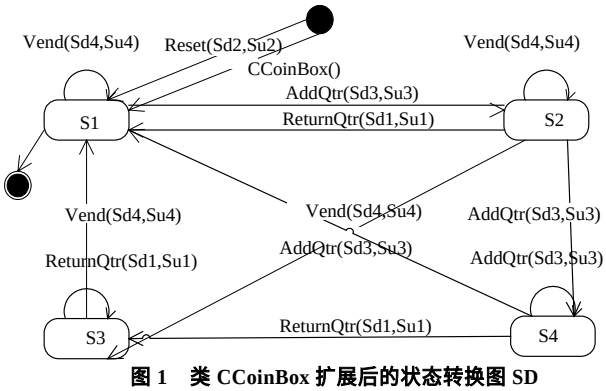

图 1　类 CCoinBox 扩展后的状态转换图 SD

根据切片标准<SD,f,v>，以成员函数 AddQtr()的变量为例，由切片工具得到<SD,AddQtr,curQtrs>的 UML 状态图切片，见图 2。

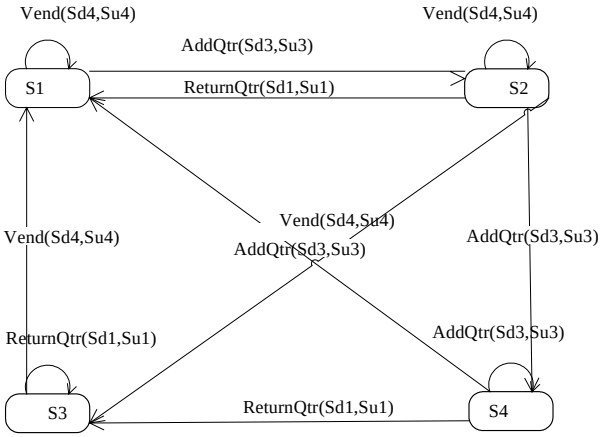

图 2 切片标准<SD,AddQtr,curQtrs>的 UML 状态图切片 SD'

在回归测试中，如果测试过程中发现成员函数 AddQtr()的错误 if 的谓词为 curQtr<1，实际上应该为 curQtr>1。我们根据 AddQtr() 的 UML 状态图切片可以得到受到这一错误可能影响到的状态。然后再去对这些可能受到影响状态进行测试和修改。

3 结束语

本文将程序切片的思想引入到对 UML 状态图的分析中，在此基础上，给出了基于依赖性分析的 UML 状态图切片的算法，并应用于回归测试中。如果按照传统的测试方法，就必须对整个程序进行重新测试。如果对一个软件进行大规模测试后，对软件进行了某种小小的修改，然后在对它重新测试显然很费时，而且效率极低。但是，借助于 UML 状态图切片，对错误的定位和排错有很好的辅助作用，可以提高测试的效率。同时，该方法对逆向工程，软件理解和维护也起到了一定的辅助作用。在将来的工作中，我们将对 UML 技术应用到软件测试中进行更深入的研究。

参考文献

1 Booch G, Rumbaugh J,Jacobson I. The Unified Modeling Language User Guide[M]. Boston:Addison-Wesley, 1999

2 Weiser M. Program Slicing[J]. IEEE Trans.Software Engineering, 1984, 16(5): 498-509.

3 Korel B, Tahat L H, Vaysburg B. Model Based Regression Test Reduction Using Dependence Analysis[C]. IEEE International Conference on Software Maintenance, 2002: 214-223.

4 Gupta R, Harrold M J, Soffa M L. An Approach to Regression Testing Using Slicing[C]. IEEE International Conference on Software Maintenance (ICSM), 1992: 299-308.

5 Larsen L D, Harrold M J. Slicing Object-oriented Software[C]. Proceedings of the 18th International Conference on Software Engineering. New York: IEEE Computer Society Press, 1996: 495-505.

6 Ferrante K, Ottenstein K, Warren J. The Program Dependence Graph and Its Use in Optimization[J]. ACM Transactions on Proramming Languages and System, 1987, 9(5): 219-349.

7 Kung D, Suchak N, Gao J, et al. On Objects State Testing[C]. Proceeding of Computer Software and Application Conference, 1994.

8 Jorgensen P C. Software Testing: A Craftsman's Approach (Second Edition)[M]. Florida: CRC Press, 2002.