基于需求的测试生成

黑盒测试

主要内容

- 引言
- 等价类划分法
- 边界值分析法

引言

- 软件需求是设计测试的基本出发点
- 通过需求设计完整有效的测试是测试团队的一项重要任务
- 需求规范可以是非形式化的，也可以是形式化的
  - 存在于人脑中或使用自然语言描述
  - 通过UML用例图、顺序图等建模元素可获得严格的需求规范
  - 使用形式化需求规约语言，如Z，RSML等，严格的需求规范可转换为形式化需求
    - 模型检测
- 需求规范的形式化程度与测试的自动化程度成正比
- 本章介绍两种黑盒测试的方法
  - 等价类划分
  - 边界值分析
- 测试用例的效率 = 测试发现的缺陷数量 / 所有缺陷
  - 需要：清晰完整的需求规范 + 细致严谨的测试用例选择策略

测试用例选择问题

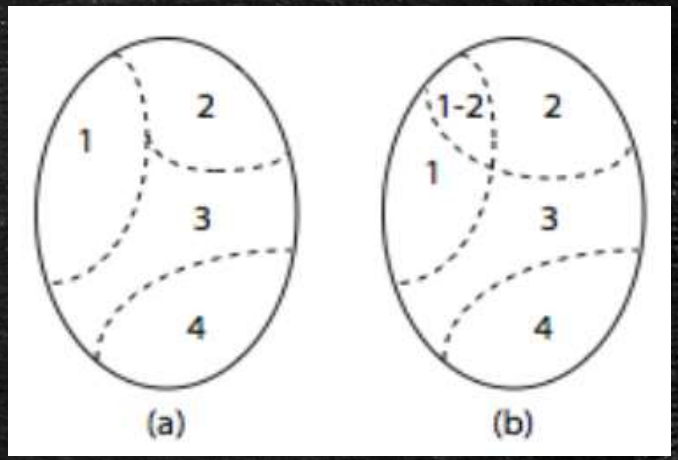
- 问题描述：设D是软件P的输入域，选取测试用例的子集T，以T中各元素为输入执行P，执行过程中将发现P中的所有缺陷
- 关键：如何构造测试用例集合$T \subseteq D$，使得T能尽可能多地发现P中的缺陷
  - 输入域的规模非常庞大，使得测试人员无法穷举所有可能的输入
  - 同时也很复杂，增加了选择测试用例的难度
- 例1：程序P的功能是对任意输入的整数序列进行升序排序。假设单个整数取值范围是$[-32768,32767]$. 所有处于该范围内的整数所构成的序列的集合构成程序P的输入域
  - 如果不限制序列长度，输入域是无限大的
  - 如果对序列长度限制为$N > 0$，则输入域的大小$S = \sum_{i=0}^{N} 65536^i$也是大到无法穷举测试

测试用例选择问题

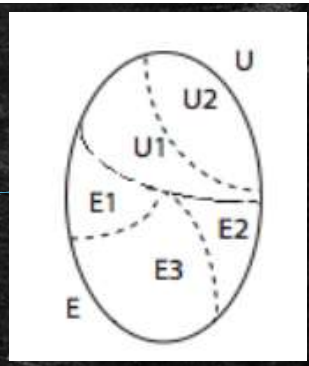
- 例2：工资管理系统程序P以雇员记录作为输入，计算雇员的周薪。雇员记录包含以下字段，每个字段都有自己的类型和约束
  - int id; // id是长度为3的数字串，取值范围为001~999
  - string name; // name是长度为20字符串，字符由26个字母或空格组成
  - float rate; // rate的取值范围是5~10美元/小时，以0.25美元的位数递增
  - int hours; // 工作小时，取值范围为0~60
- 输入域为所有可能雇员记录，其大小为$999 \times 27^{20} \times 21 \times 61$
- 测试用例选择方法：从输入域中选择一个尽可能小的子集，达到尽可能全面测试的目的

等价类划分概述

- 将输入域划分为数量尽可能少的若干子域。子域两两互不相交。右图中(a)将输入域划分为4个子域，每个子域构成一个等价类。图(b)不构成划分，因为子域1和2有交集
  - 回想离散数学中的等价关系和等价类划分的概念
- 划分原则：用同一等价类中的任意输入对软件进行测试，软件都输出相同的结果。在这种原则下，只需从每个等价类中选取一个输入作为测试用例即可。最终测试用例集的大小跟划分后等价类或子域的个数相同
- 对同一输入域，可采用不同的划分方法，得到的划分可能不同，从而最终的测试用例集也不同
- 即使划分后的等价类相同，也可能会选择不同的测试用例，从而最终的测试用例集也不同



缺陷定位



- 输入域至少可分为两个子集：合法的子集和非法的子集，分别记为E和U
  E和U又可以继续划分为若干子集
- 等价类划分法就是要从E和U及其子集中选择适当的输入作为测试用例，以便发现软件当中存在的错误
- 例：在软件S中，表示人员年龄的变量为整型，其合法值应在[1,120]范围内
  - $E = \{i | 1 \leq i \leq 120\}$，$U = \{i | i < 1 \ or \ i > 120\}$
  - 假设需求R_1处理所有取值在[1,60]之间的输入，而需求R_2处理取值在[61,120]之间的输入。可以将E划分为两个子集
  - U也可以划分为两个子集，一个表示小于1的取值，一个表示大于120的取值
  - 这样就得到四个子域，每个子域对应一个需求
  - 可从这四个子域中各取一个输入构成测试用例集，每个输入可发现对应需求中的缺陷

关系和等价类划分

- 集合论中，关系是一个n元组的集合
- 每个软件，程序，方法或函数都定义了一个输入域和输出域之间二元关系
- 例：函数addList计算并返回一个整数序列之和
  - 二元关系$addList$: $L \to Z$: 如((1,2),3), ((1,3,5,7,9),25), (( ), 0)等
  - 如果函数addList存在错误：$addList: L \to Z \cup \{error\}$
- 划分输入域时采用的关系为$R: I \to I$.
  - R是I上的关系，它定义了一个等价类，I为输入域

定义： 设 R 为定义在集合 A 上的等价关系，对于 $a \in A$，集合 $[a]_R = \{x | x \in A, aRx\}$ 称为元素 a 形成的 R **等价类**，简称**等价类**。

定义：任一序偶的集合确定了一个**二元关系** R，该集合中的任一序偶 $\langle x,y\rangle$ 可记作 $\langle x,y\rangle \in R$ 或 xRy；不在该集合中的序偶 $\langle x,y\rangle$ 可记作 $\langle x,y\rangle \notin R$ 或 $x\not Ry$。

定义：设 R 为定义在集合 A 上的二元关系，若 R 是自反的、对称的、传递的，则称 R 为**等价关系**。

等价关系的例子很多，如三角形的全等、相似，方阵的相似，整数集上的模 k 同余，都是等价关系。

定义：设 R 为定义在集合 A 上的等价关系，对于 $a \in A$，集合 $[a]_R = \{x|x \in A, aRx\}$ 称为元素 a 形成的 R **等价类**，简称**等价类**。

例如，定义在整数集 $\mathbb{Z}$ 上的关系 R 代表模 3 同余，那么由 $\mathbb{Z}$ 的元素所产生的等价类是

$$\ldots = [-3]_R = [0]_R = [3]_R = \ldots = \{x|x = 3k, k \in \mathbb{Z}\},$$
$$\ldots = [-2]_R = [1]_R = [4]_R = \ldots = \{x|x = 3k+1, k \in \mathbb{Z}\},$$
$$\ldots = [-1]_R = [2]_R = [5]_R = \ldots = \{x|x = 3k+2, k \in \mathbb{Z}\}。$$

关系和等价类划分（二）

- 例：接收离散的输入，且对任意有效输入的处理方式都相同，此时给出有效和无效两个等价类即可
  - 需求：函数getPrice以食品杂货店的食品名称作为输入，查询商品价格数据库并返回相应食品的价格。如果数据库中没有该食品，返回错误信息“Not Found”
  - 输入域：所有可能的食品名称字符串，如milk, tomato, ……
  - 在输入域上定义等价关系：$pFound: I \to I$。数据库中存在的食品均属于同一等价类$pFound$，而不存在的食品均属于另一等价类$pNFound$. $pFound \cup pNFound = I, pFound \cap pNFound = \emptyset$

关系和等价类划分（三）

- 例：接收离散输入，程序运行方式依赖于具体的输入值，这些输入值可分为若干类
  - 需求：打印机自动测试软件$pTest$以打印机品牌和型号作为输入（通过键盘输入），从测试脚本列表中选取相应的测试脚本，然后执行脚本，验证打印机功能是否正常。该软件根据输入的打印机类型来选择相应脚本。假设有3种类型的打印机：彩色喷墨打印机（ci），彩色激光打印机（cl），彩色多功能打印机（cm）。如果输入的是HP Deskjet 6840，$pTest$将选择彩色喷墨打印机的脚本。设计测试用例验证脚本选择功能是否正确
  - $pTest$的输入域I由表示打印机品牌和型号的字符串构成，既包含有效的字符串，也包含无效的字符串。
  - 定义四个关系：$ci: I \to I, cl: I \to I, cm: I \to I, invP: I \to I.$
    - 关系cl将所有彩色激光打印机划分为一个等价类，将其他打印机划分为另一个等价类
    - $invP$表示无效输入的等价关系
    - 每个关系对应两个等价类，一共8个等价类，但是相互之间有重叠，不能构成划分
  - 定义$pCat$，根据ci，cl，cm，$invP$ 4个类别将$pTest$的输入域划分为4个等价类

关

- 在
  划
  -
  -
  -

等价类	w	f
E1	非空串	存在，非空文件
E2	非空串	不存在
E3	非空串	存在，空文件
E4	空串	存在，非空文件
E5	空串	不存在
E6	空串	存在，空文件

```
1 begin
2    string w, f;
3    input (w, f);
4    if (¬ exists(f)) {raise exception; return(0)};
5    if (length(w)==0)return(0);
6    if (empty(f)) return(0);
7    return(getCount(w, f));
8 end
```

关系和等价类划分（五）

- 前面的例子都是从程序的输入输出等价关系。有些情况下，可以从程序的输出导出等价关系。或者将两者进行结合

变量等价类划分的基本原则

表 2-1　取值范围（range）和字符串（string）变量的等价类划分原则

类 别	等价类	示　例	
		约 束	等价类代表①
取值范围（range）	一个取值范围内的等价类；两个取值范围外的等价类	speed∈[60,90]	{{50}↓,{75}↑,{92}↓}
		area:float;	{{-1.0}↓,{15.52}↑}
		area≥0	
		age:int;	{{-1}↓,{56}↑,{132}↓}
		0≤age≤120	
		letter:char;	{{J}↑,{3}↓}
字符串（string）	至少分为一个包含所有合法字符串的类和一个包含所有非法字符串的类。合法性由字符串的长度及其他语义特性所决定	fname:srting;	{{ε}↓,{Sue}↑,{Sue2}↓,{Too Long a name}↓}
		vname:string;	{{ε}↓,{shape}↑,{address1}↑,{Long variable}↓}

① 每个等价类后的符号：↓非法输入等价类的代表，↑合法输入等价类的代表。

张秀

幻灯片 14

张秀峰2 张秀峰, 2018/2/25

变量等价类划分的基本原则

表 2-2　枚举（enumeration）和数组（array）变量的等价类划分原则

类 别	等价类	示　例①	
		约 束	等价类代表②
枚举 (enumeration)	每个取值对应一个等价类	auto_color ∈ {red, blue, green} up: boolean	{{red} ↑, {blue} ↑, {green} ↑} {{true} ↑, {false} ↑}
数组 (array)	一个包含所有合法数组的等价类，一个空数组等价类，以及一个包含所有大于期望长度数组的等价类	Jave array: int [] aName = new int [3]	{{[ ]} ↓, {[-10,20]} ↑, {[-9,0,12,15]} ↓}

① 参见对不同项的解释。

② 每个等价类后的符号：↓非法输入等价类的代表，↑合法输入等价类的代表。

变量等价类划分的基本原则

- 复合数据类型
  - 先对内部简单类型进行等价类划分
  - 对各变量的等价类进行组合

程序 P2.2

```
1 struct R
2   {
3     string fName; //名
4     string lName; //姓
5     string cTitle [200]; //课程名称
6     char grades [200]; //课程成绩
7   }
```

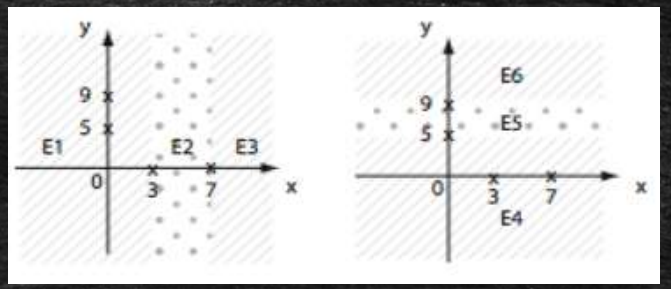
一元划分和多元划分

- 一元划分：每次只考虑一个变量，每个输入变量形成对输入域的一个划分。有几个变量，就有几种划分。每种划分包含两个或多个等价类
- 多元划分：将所有输入变量的笛卡尔积作为输入域，并定义该域上的等价关系。最后只产生一种划分
- 测试用例的选择一般使用一元划分，优点是简便好管理
- 多元划分产生的等价类数量大，人工管理困难，并且有可能存在没用的等价类。但是其优点是：比一元划分测试得更充分

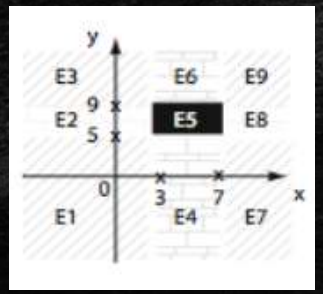
一元划分和多元划分

- 例：某程序的输入为整型数据x和y，其取值范围分别为[3,7]和[5,9]。
  - 一元划分，可得6个等价类

E1: x<3	E2: 3≤x≤7	E3: x>7
E4: y<5	E5: 5≤y≤9	E6: y>9



  - 多元划分



E1: x<3, y<5

E3: x<3, y>9

E2: x<3, 5≤y≤9

E4: 3≤x≤7, y<5

E5: 3≤x≤7, 5≤y≤9

E6: 3≤x≤7, y>9

E7: >7, y<5

E8: x>7, 5≤y≤9

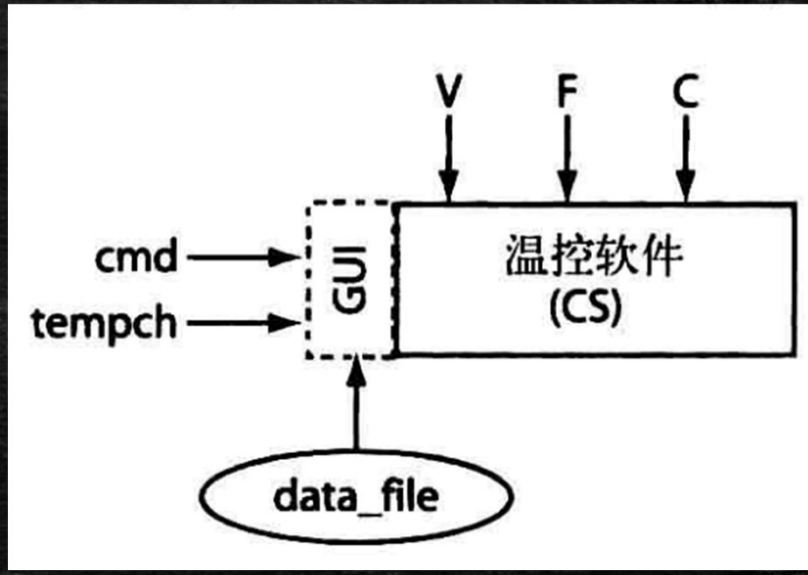
E9: x>7, y>9

等价类划分的完整过程

- 不论软件规模大小，都可以使用等价类划分法设计测试用例
- 但如果变量比较多，人工方式可能会存在困难，最好使用辅助工具
- 步骤1：确定输入域
  - 分析并确定需求中的所有输入和输出变量，变量类型，使用条件，
  - 环境变量（如OS类型）可作为输入变量
  - 参考需求中对变量的约束，确定各变量的取值集合
- 步骤2：等价类划分
  - 将每个变量的取值划分为互不相交的子集，每个子集对应一个等价类，从而构成对输入域的划分
  - 划分时，将具有相同处理方式的变量取值放到一个等价类中
- 步骤3：组合等价类（可选）
  - 假设变量X的等价类子集为$\{X_1, X_2\}$，Y的等价类子集为$\{Y_1, Y_2, Y_3\}$，组合后的等价类$E = \{X_1 \times Y_1, X_1 \times Y_2, X_1 \times Y_3, X_2 \times Y_1, X_2 \times Y_2, X_2 \times Y_3\}$
  - 组合爆炸问题

等价类划分的完整过程

- 步骤4：确定不可测的等价类
  - 有些输入数据实际测试过程中无法输入到系统中，包含这类数据的等价类称为不可测等价类
    - 例：某软件的数据只有通过GUI才能输入，而GUI中只包含了有效的数据，不包含无效的数据。因此前述步骤中生成的无效数据的等价类是不可测等价类
- 例：为热水器温控软件设计测试用例
  - 需求：热水器温控系统简称BCS。BCS的温控软件简称CS
    - 操作员使用的控制选项C包含3个命令（cmd）：温度控制命令（temp）、系统关闭命令（shut）、请求取消命令（cancel）
    - 操作员选择C后，BCS检查V，如果V为GUI，则操作员通过GUI选择一条命令（cmd）执行。如果V为文件（file），BCS通过命令文件获取命令(cmd)执行。命令文件名由变量F表示
    - 命令temp要求操作员输入温度调节数值tempch，可选有效值为-10，-5，5，10，这4个值 记为t_valid，其余的值记为$t_invalid$



等价类划分的完整过程

- 确定输入域
- 输入域$I = V \times F \times cmd \times tempch \cup \{file, cmd_file, temp, t_invalid\}$

变量	种类	类型	取值
V	环境变量	枚举	{GUI，*file*}
F	环境变量	字符串	文件名
cmd	GUI 或文件方式输入	枚举	{*temp*，*cancel*，*shut*}
tempch	GUI 或文件方式输入	枚举	{-10，-5，5，10}

等价类划分的完整过程

- 等价类划分

变量	等价类划分
V	{{GUI}, {*file*}, {*undefined*}}
F	{*f_valid*}, {*f_invalid*}
cmd	{{*temp*}, {*cancel*}, {*shut*}, {*c_invalid*}}
tempch	{−10}, {−5}, {5}, {10}, {*t_invalid*}

等价类划分的完整过程

- 组合等价类
  - 共3*2*4*5=120个等价类
  - {(GUI,f_valid,temp,-10)}
  - {(GUI,f_valid,temp,-5)}
  - {(GUI,f_valid,temp,t_invalid)}
- 去除不可测等价类
  - 只有temp命令才需要tempch: 去掉所有的$\{(_,_,\{cmd\}\setminus\{temp\},_)\}$: （3*2*3*5=90）
  - 在GUI方式下，不允许非法tempch值，再去掉2个不可测等价类
  - 当V为undefined时，不需要读取cmd和tempch的具体值，再去掉5个不可测等价类
  - 当F为无效文件名时，不需要获取cmd和tempch的具体值，再去掉5个不可测等价类
- 最后还剩（120-90-2-5-5=18）个等价类（还有问题）
  - $\{(GUI, f\_valid, temp, t\_valid)\}$ (4个)，$\{(GUI, f\_invalid, temp, t\_valid)\}$（4个）
  - $\{(GUI, \_, cancel, NA)\}$（2个），$\{(GUI, \_, shut, NA)\}$（2个）
  - $\{(file, f\_valid, temp, t\_valid \cup t\_invalid)\}$（5个），$\{(file, f\_invalid, NA, NA)\}$
  - $\{(undefined, NA, NA, NA)\}$ （1个）

边界处的错误

- 经验表明程序员在处理等价类边界附近时容易出错
- 例：当$x \leq 0$时，方法M需要计算函数$f1$，否则计算$f2$。方法M的一个错误就在于程序员将$x \leq 0$写成了$x < 0$
- 此时，将$x = 0$作为测试用例运行M，可发现这一错误。但是，如果测试用例来自于等价类时，比如$x \in \{-4,7\}$，就不会发现这个错误。此时$x = 0$正好处于两个等价类$x \leq 0$和$x > 0$的边界上。

边界值分析（Boundary Value Analysis）

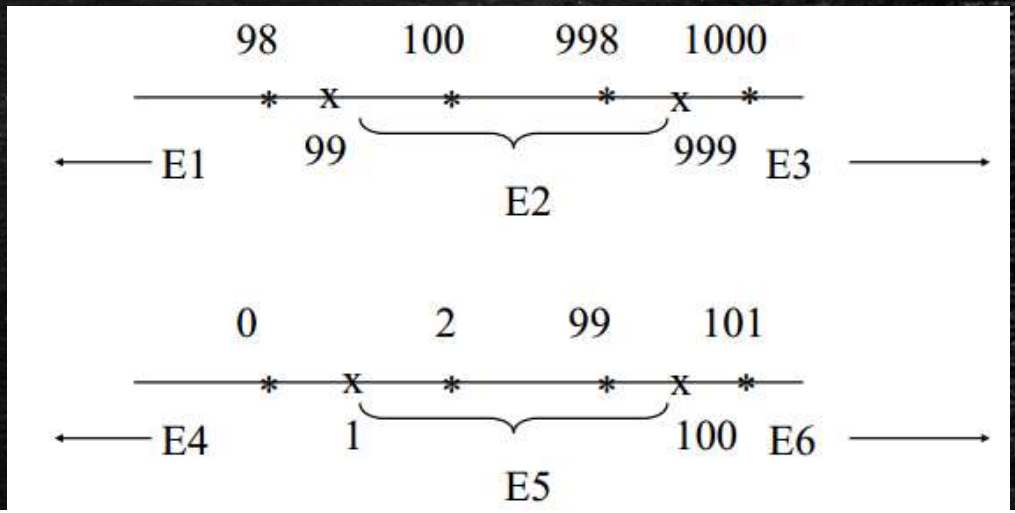
- 边界值分析（BVA）是一种有效的测试用例选择方法，可以发现位于等价类边界上的错误
- 边界或边界附近选取测试用例。等价类划分法从等价类中选取测试用例，而边界值法从等价类
- 两者之间可能重叠

主要步骤

- 步骤1：使用一元划分法划分输入域。有多少个变量就有多少种划分
- 步骤2：为每种划分确定边界。也可利用输入变量之间的特定关系确定边界
- 步骤3：设计测试用例，确保每个边界值至少出现在一个测试输入数据中

例子

- 需求：函数fP有两个整型输入变量$code$和qty，分别表示商品编码和采购数量。fP访问数据库查询$code$编码对应产品的单价，描述信息和总的采购价格。当$code$和qty中任意一个为非法输入时，显示错误信息
- 为两个变量创建等价类。假设$code$变量的有效输入区间为[99,999]，采购数量qty的有效输入区间为[1,100]。得到如下等价类：
  - code变量的等价类：E1：<99，E2：有效区间取值，E3：>999
  - qty变量的等价类：E4：<1，E5：有效区间取值，E6：>100
- 确定边界：
  - *和x标识
  - 共12个，每个变量6个



例子（续）

- 设计测试用例
  - t1和t6中两个变量均取非法值，无法发现只检查一个变量合法性的错误
  - 需要添加新测试用例
    - t7:(code=98,qty=45)
    - t8:(code=1000,qty=45)
    - t9:(code=250,qty=0)
    - t10:(code=250,qty=101)
- T={t2,t3,t4,t5,t7,t8,t9,t10}
- t2和t5同时出现两个变量的边界值，有什么

T={ t1: (code=98, qty=0),
t2: (code=99, qty=1),
t3: (code=100, qty=2),
t4: (code=998, qty=99),
t5: (code=999, qty=100),
t6: (code=1000, qty=101)
}

- If(code <99 && qty < 1) {
- 报错
- }

ATM问题

货币转换器

雨刷控制器