第六章面向源代码的软件可信性度量模型

陈仪香

2024年11月11日

Outline

1. 6.1 经验系统
2. 6.2 数值系统
3. 6.3 测量映射
4. 6.4 Extensive结构
5. 6.5 软件度量模型
6. 6.6 软件可信性规范

第六章面向源代码的软件可信性度量模型

- 度量理论提供了一种用数值来刻画实体属性直觉性质的框架。
- 本章将度量理论应用于软件可信性度量中，基于度量理论中Extensive结构建立面向源代码的软件可信性度量模型，并利用公理化方法对所建软件可信性度量模型合理性进行理论验证，从而得到暨经度量理论验证又经公理化方法验证的软件可信性度量模型。
- 通过一个简单案例表明该度量模型的可用性和有效性。

6.1 经验关系系统

经验关系系统

Definition (经验关系系统)

给定一个属性，经验关系系统是一个有序元组

$$ERS = (E, R_1, \cdots, R_m, o_1, \cdots, o_n)$$

其中，

1. E：度量其属性的实体非空集；
2. $R_i(1 \leq i \leq m)$：集合E上的k_i元经验关系，用来表示关于属性的直觉知识；
3. $o_j(1 \leq j \leq n)$：集合E上的二元运算，即$o_j : E \times E \to E$。

经验关系系统

比如，假定我们要度量程序，则建立关于程序的经验关系系统

$$ERS_P = (E, \gg, ;)$$

其中，

1. E 为所有程序组成的集合；
2. $\gg \subseteq E \times E$表示关于程序经验理解的二元关系：给定两个程序e_1和e_2，若程序e_2是程序e_1的子程序，则我们认为$e_1 \gg e_2$，
3. 二元运算；：$e_3 = e_1; e_2$表示程序e_1和e_2的顺序连接。

经验关系系统

- 同一个实体集的其它属性，比如，复杂性，又不同于长度所具有的经验关系和操作，这是因为对于不同属性我们有不同经验理解。
- 经验关系系统只是用来人们关于实体属性的经验理解，而没有数或值的概念，因此也没有度量概念。
- 下面引入数值关系系统，用来进行度量引入。

6.2 数值关系系统

数值关系系统

Definition (数值关系系统)

给定一个属性，数值关系系统是一个有序元组

$$NRS = (V, S_1, \cdots, S_m, \oplus_1, \cdots, \oplus_n)$$

其中，

1. V：度量属性的度量值集；
2. $S_i(1 \leq i \leq m)$：集合V上的k_i元经验关系；
3. $\oplus_j(1 \leq j \leq n)$：集合$V$上的二元运算，即$\oplus_j$：$V \times V \to V$。

数值关系系统中V的元素一般是通常意思上的非负实数，但有时也可以是由一些A、B、C、D之类的符号组成。

数值关系系统

比如，在上面提到的程序长度例子中，我们可以定义其程序长度数值关系系统

$$NRS_{PL} = (V_I, \geq, +)$$

其中，

1. V_I是所有正整数组成的集合；
2. 二元关系为整数的普通大小关系$\geq$；
3. 二元操作为整数间普通加法$+$。

数值关系系统

再比如，定义有关程序的缺陷数值关系系统$NRS_{PE} = (V_e, \sqsubseteq, \wedge)$，其中，

- V_e是程序分类集合$\{A, B\}$，A表示程序无缺陷，而B表示程序有缺陷；
- V_e上的二元关系$\sqsubseteq$ 定义为$A \sqsubseteq B$，表示有缺陷程序的缺陷程度是大于等于无缺陷程序的缺陷程度；
- $\wedge$是集合V_e上的二元运算：

$$A \wedge A = A, B \wedge B = B, A \wedge B = B$$

表示有缺陷程序与任何程序合并都是有缺陷的。

数值关系系统的建立是为了将经验关系系统反映到值上，因此需要建立从经验关系系统到数值关系系统的联系，即，测量映射。

6.3 测量映射

测量映射

Definition (测量映射(Measure))

设$ERS = (E, R_1, \cdots, R_m, o_1, \cdots, o_n)$是一个经验关系系统，$NRS = (V, S_1, \cdots, S_m, \oplus_1, \cdots, \oplus_n)$是一个数值关系系统。任何一个从$E$到$V$的函数$\mu : E \to V$ 都称为测量映射。

- 对于程序经验关系系统ERS_P和程度数值关系系统NRS_{PL}，我们定义一个测量函数$\mu_L : E \to V_I$：给定一个程序e，$\mu_L(e)$ =程序e的代码行数。
- 同样，对于程序缺陷数值关系系统NRS_{PE}, 我们也可以定义一个测量函数$\mu_D : E \to V_e$：给定一个程序e，若程序e没有缺陷则$\mu_D(e) = A$ 否则 $\mu_D(e) = B$，即A表示程序没有缺陷，而B表示程序有缺陷。

测量映射

测量映射只考虑从E到V的映射，而没有考虑从经验关系系统中经验关系和经验操作到数值关系和数值操作之间的映射，范围过于宽泛，容易出现度量结果和人们经验理解不一致的地方，为此引入标度概念。

标度(Scale)

Definition (标度(Scale))

令$ERS = (E, R_1, \cdots, R_m, o_1, \cdots, o_n)$ 为一个经验关系系统，$NRS = (V, S_1, \cdots, S_m, \oplus_1, \cdots, \oplus_n)$ 为一个数值关系系统，$\mu : E \to V$ 是一个测量映射，则称(ERS, NRS, μ) 为一个标度，若对于经验关系系统的k元经验关系R_i和数值关系系统的k元经验关系$S_i (1 \leq i \leq m)$ 都有对于所有的$j (\in \{1, \cdots, n\})$ 和所有的$a_1, \cdots, a_k$，b，$c \in E$ 下式成立

$$R_i(a_1, \cdots, a_k) = S_i(\mu(a_1), \cdots, \mu(a_k))$$

以及

$$\mu(b\ o_j\ c) = \mu(b) \oplus_j \mu(c)$$

刻度：例子

- 程序经验关系系统ERS_P、程序长度数值关系系统NRS_{PL}和测量映射μ_L构成的$(ERS_P,\ NRS_{PL},\ \mu_L)$是一个标度，
- 这是因为给定两个程序$e1$ 和$e2$，若$e1 \gg e2$ 则$e2$ 是$e1$的子程序，从而$e1$的长度大于等于$e2$的长度，即$\mu_L(e1) \geq \mu_L(e2)$，
- 同时$e1;e2$的长度等于$e1$长度与$e2$长度之和，即$\mu_L(e1;e2) = \mu_L(e1) + \mu_L(e2)$。

刻度：例子

- 程序经验关系系统ERS_P、程序缺陷数值关系系统NRS_{PE}和测量映射μ_D构成的(ERS_P, NRS_{PE}, μ_D)也是一个标度，
- 这是因为给定两个程序$e1$和$e2$，若$e1 \gg e2$ 则$e2$ 是$e1$的子程序，若$e2$有缺陷则$e1$一定有缺陷，即若$\mu_D(e2) = B$ 则$\mu_D(e1) = B$，
- 同时即使$e2$没有缺陷也不能判断$e1$没有缺陷，即在$\mu_D(e2) = A$的前提下，$\mu_D(e1) = A$或B，因此有$\mu_D(e1) \sqsupseteq \mu_D(e2)$;
- 另外，若$e1$和$e2$中有一个有缺陷，则$e1; e2$ 一定有缺陷，进而得到$\mu_D(e1; e2) = \mu_D(e1) \wedge \mu_D(e2)$。

6.4 Extensive结构

Extensive结构

Extensive结构是一种特殊经验关系系统，由度量对象集A、度量对象上的二元关系R、度量对象上一个二元运算$\circ$三部分组成，其中

- 二元关系满足强完备性、传递性；
- 二元闭操作需满足弱结合律、单调性和阿基米德性。

物理科学中关于诸如长度、体积等的度量都是基于该结构。目前该结构也用于程序复杂性等内部属性的度量中.

Extensive结构

Definition (Extensive 结构)

一个经验关系系统$(A，R，\circ)$称为具有Extensive结构，若下列各条成立：

1. 弱序性：$(A，R)$ 是一个弱序，即R满足传递性和强完备性，强完备性是指对$\forall a，b \in A$，有$(a，b) \in R$或$(b，a) \in R$;
2. 弱结合律：$\forall a，b，c \in A$：$(a \circ b) \circ c \approx a \circ (b \circ c)$;
3. 单调性：$\forall a，b，c \in A$：$(a，b) \in R$ 当且仅当 $((a \circ c)，(b \circ c)) \in R$当且仅当 $((c \circ a)，(c \circ b)) \in R$;
4. 阿基米德性：如果$\forall a，b，c，d \in A$： 如果$(a，b) \in R_s$，则存在正整数 n 使得$((na \circ c)，(nb \circ d)) \in R_s$成立。

其中，

$$a \approx b \overset{\text{定义}}{=\!=} (a,b) \in R \text{ 并且 } (b,a) \in R$$

$$(a,b) \in R_s \overset{\text{定义}}{=\!=} (a,b) \in R \text{ 但是 } (b,a) \notin R$$

$1a = a$ 归纳定义 $(n+1)a = na \circ a$

Extensive结构表示和唯一性定理

Theorem (Extensive结构表示和唯一性定理)

设$(A, \succeq, \circ)$是一个经验关系系统，$\Re$是所有实数组成的集合。则存在(实值)测量函数$\mu: A \to \Re$ 使得对于任意$a, b \in A$使下式成立

$$a \succeq b \Leftrightarrow \mu(a) \geq \mu(b) \text{ 及 } \mu(a \circ b) = \mu(a) + \mu(b)$$

当且仅当$(A, \succeq, \circ)$是具有*Extensive* 结构。
如果另外一个(实值)测量函数μ'满足上式，则存在一个正实数r使得

$$\mu' = r\mu$$

成立。

Extensive结构表示和唯一性定理

- 这个定理说：具有Extenvie结构的经验关系系统一定有一个实值数值关系系统$(\Re, \geq, +)$，并且在实数倍的前提下是唯一的，即若有两个实值数值关系系统，则它们的测量函数相差实数倍。
- 我们经常接触到的物理科学中关于诸如长度、体积、时间等的度量都是基于Extensive结构。

Extensive结构表示和唯一性定理

- 程序经验关系系统NRS_P是具有Extensive结构，并且存在一个到程序长度数值关系系统$NRS_{PL} = (V_I, \geq, +)$的实值测量函数$\mu_L$ 将程序e的行数指定给程序e，即$\mu_L(e)$ =程序e的行数。
- 目前该结构应用于程序复杂性等内部属性度量中，但尚未被用于外部属性以及软件可信性度量中，主要原因是针对某一外部属性或者软件可信性难于构建满足弱序等一系列公理并被大家所接受的度量对象间经验关系，从而也就难于构建相应的Extensive结构。

6.5 面向源代码的软件可信性度量模型

6.5 面向源代码的软件可信性度量模型

本节选择基于Extensive结构进行软件可信性度量。

- 需建立用于刻画度量对象集、度量对象间经验关系和度量对象间操作的经验关系系统；
- 构造用于描述度量对象值集、值间关系以及值间操作的数值关系系统；
- 构造一个从经验关系系统到数值关系系统的测量函数，该函数把度量对象、对象间经验关系和对象间操作分别映射到数值关系系统中相对应的值、值间关系和值间操作。这个测量函数还应该是从经验关系系统到数值关系系统的同态映射。
- 本节接下来工作主要围绕这几个方面展开。

软件可信性经验关系系统

- 要对度量对象进行度量，首先必须给出它们的抽象表示，这样才可以对它们进行分析，对它们相关属性进行量化。本章是从源代码角度对软件可信性进行度量，因此下面所给出的抽象模型是关于源程序的抽象表示。
- 我们将整个程序看做一个系统，系统是由各个不同模块通过控制依赖或者数据依赖等关系连接在一起得到，而模块本身又是一个更小的系统，又有相应的程序元素并通过一定的关系进行相关联。
- 程序元素，简称为元素，包括变量、常量、表达式、各种类型语句和程序单元。
- 程序单元，简称单元，是一个实现特定目的可以被调用的代码，像C语言中的函数，Java语言中的方法等。

6.5 面向源代码的软件可信性度量模型

Definition (系统)

软件软件系统S为一个二元组(E, R)，其中E是S中程序元素组成的集合，R是E上一个二元关系，即$R \subseteq E \times E$。

Definition (模块)

给定一个系统$S = (E, R)$，称系统$m = (E_m, R_m)$为S的一个软件模块，简称模块，若

$$(E_m \subseteq E) \wedge (R_m \subseteq R)$$

成立。

比如E可以是程序中的语句或者程序块组成的集合，而R可以是这些语句或者程序块之间的控制流图，一个模块$m = (E_m, R_m)$可以是代码片段或子程序。
最好一个模块实现一个功能。即，按照功能划分模块。

C语言编写的快速排序程序QS

```c
#include <stdio.h>
int Partition(int *R,int i,int j)
{
    int pivot=R[i];
    while(i<j)
    {
        while(i<j&&R[j]>=pivot)
            j--;
        if(i<j)
            R[i++]=R[j];
        while(i<j&&R[i]<=pivot)
            i++;
        if(i<j)
            R[j--]=R[i];
    }
    R[i]=pivot;
    return i;
}
void QuickSort(int *R,int low,int high)
{
    int pivotpos;
    if(low<high)
    {
        pivotpos =Partition(R,low,high);
        QuickSort(R,low,pivotpos-1);
        QuickSort(R,pivotpos+1,high);
    }
}//QuickSort
int main()
{
    int s[100];
    printf("input␣10␣number");
    for(int i=0;i<10;i++)
    {
        scanf("%d",&s[i]);
    }
    QuickSort(s,0,9);
    for (int j=0;j<10;j++)
    {
        printf("%d␣",s[j]);
    }
}
```

给定一个程序系统$S=(E，R)$，令MS表示系统S中所有模块组成的集合，则MS为我们所要构建的经验关系系统的度量对象集。

Definition (模块二元运算$\circ_T$)

设$m_1=(E_{m_1}，R_{m_1}), m_2=(E_{m_2}，R_{m_2})\in MS$，定义模块$E_{m_1}$和$E_{m_2}$间的二元关系$R_{m_1m_2}$:

$$R_{m_1m_2}=\{(e_1，e_2)\mid e_1\in E_{m_1}\wedge e_2\in E_{m_2}\wedge(e_1，e_2)\in R\}$$

则两模块间的二元操作$\circ_T$定义如下

$$m_1\circ_T m_2=(E_{m_1}\uplus E_{m_2}，R_{m_1}\cup R_{m_2}\cup R_{m_1m_2})$$

其中，$E_{m_1}\uplus E_{m_2}$表示由E_{m_1}和E_{m_2}中所有元素组成的多重集，即若程序单元e在两个模块E_{m_1}出现n_1次在模块E_{m_2}中出现n_2次，则在e在模块$E_{m_1}\uplus E_{m_2}$出现n_1+n_2次。这样程序单元在一个模块中可能出现多次。由模块定义得$m_1\circ_T m_2\in MS$，即$\circ_T$是模块间的二元闭操作，即二元运算。

C语言编写的快速排序程序QS

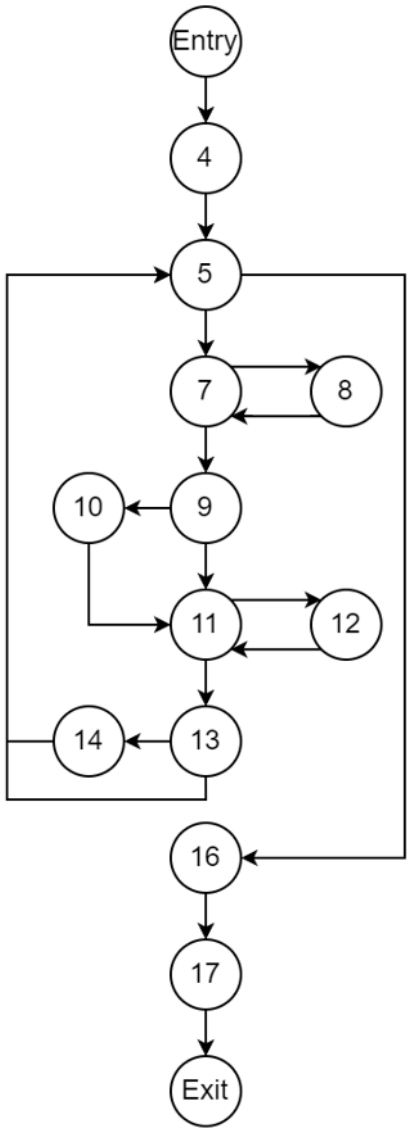
```c
#include <stdio.h>
int Partition(int *R,int i,int j)
{
    int pivot=R[i];
    while(i<j)
    {
        while(i<j&&R[j]>=pivot)
            j--;
        if(i<j)
            R[i++]=R[j];
        while(i<j&&R[i]<=pivot)
            i++;
        if(i<j)
            R[j--]=R[i];
    }
    R[i]=pivot;
    return i;
}
void QuickSort(int *R,int low,int high)
{
    int pivotpos;
    if(low<high)
    {
        pivotpos =Partition(R,low,high);
        QuickSort(R,low,pivotpos-1);
        QuickSort(R,pivotpos+1,high);
    }
}//QuickSort
int main()
{
    int s[100];
    printf("input␣10␣number");
    for(int i=0;i<10;i++)
    {
        scanf("%d",&s[i]);
    }
    QuickSort(s,0,9);
    for (int j=0;j<10;j++)
    {
        printf("%d␣",s[j]);
    }
}
```

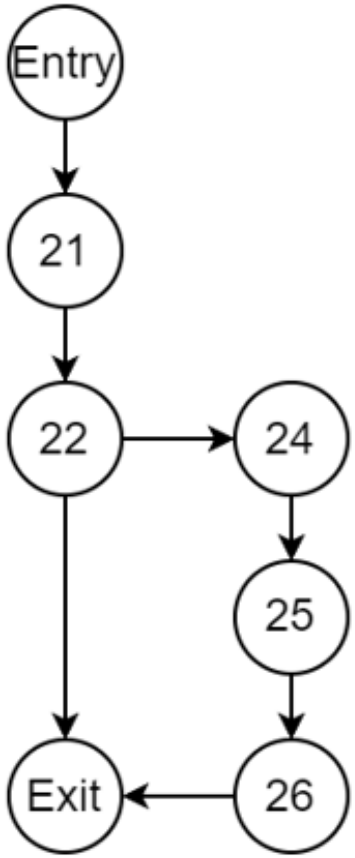
C语言编写的快速排序程序QS

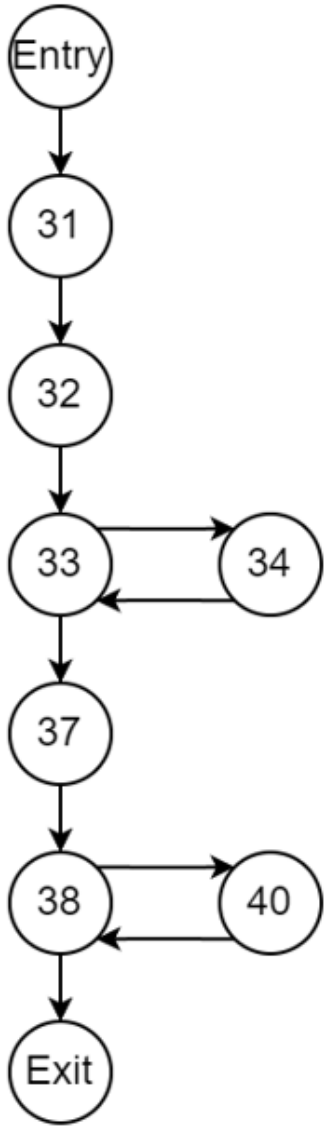
这个软件含有42行代码，第一行是头文件，后面各模块都要使用它，因而不记载模块中。
将这个代码分成三个模块：令$M = \{m_1 =< E_{m_1}, R_{m_1} >, m_2 =< E_{m_2}, R_{m_2} >, m_3 =< E_{m_3}, R_{m_3} >\}$，其中

- E_{m_1}由第2-18行中语句组成的集合，
- E_{m_2} 由第19-28行中语句组成的集合，
- E_{m_3} 由第29-42行中语句组成的集合。

C语言快速排序程序QS:模块M1、M2、M3的控制流图







6.5 面向源代码的软件可信性度量模型

- 软件可信性是软件行为和结果符合用户预期的能力，无论是软件行为还是运行结果都由其实现机制所决定。
- 因此，给定一个软件，其可信性可通过求解软件本身所采用实现机制和用户期望所采用实现机制的相似程度来建立；
- 同时软件本身又是由模块通过控制或数据依赖等关系组成的集合，这样软件可信性度量又可转化为其模块所采用实现机制和用户期望模块所采用实现机制的相似程度来计算。
- 另一方面，模块是由程序元素依据一定关系组成的集合，从而软件可信性度量又可进一步通过计算程序元素所采用实现机制和用户期望程序元素所采用实现机制的相似程度来得到。

6.5 面向源代码的软件可信性度量模型

- 给定一个程序系统$S=(E,R)$, MS是S的模块集合，$E_m \in MS$。
- 符号$\#(E_m)_{expect}$表示观察到模块m中具有用户期望机制实现程序元素个数，$\#(E)$ 表示E 中所有程序元素个数。
- 先引入符号$\kappa(m)$ 表示观察到模块$m=(E_m,R_m)$中具有用户期望机制实现程序元素个数与E中所有程序元素个数的比值，即

$$\kappa(m)=\frac{\#(E_m)_{expect}}{\#(E)}$$

Claim

给定两个模块$m_1=(E_{m_1},\ R_{m_1})$，$m_2=(E_{m_2},\ R_{m_2}) \in MS$，则$\kappa(m_1 \circ_T m_2)=\kappa(m_1)+\kappa(m_2)$。

按照模块二元运算定义进行证明。

C语言编写的快速排序程序QS

```c
#include <stdio.h>
int Partition(int *R,int i,int j)
{
    int pivot=R[i];
    while(i<j)
    {
        while(i<j&&R[j]>=pivot)
            j--;
        if(i<j)
            R[i++]=R[j];
        while(i<j&&R[i]<=pivot)
            i++;
        if(i<j)
            R[j--]=R[i];
    }
    R[i]=pivot;
    return i;
}
void QuickSort(int *R,int low,int high)
{
    int pivotpos;
    if(low<high)
    {
        pivotpos =Partition(R,low,high);
        QuickSort(R,low,pivotpos-1);
        QuickSort(R,pivotpos+1,high);
    }
}//QuickSort
int main()
{
    int s[100];
    printf("input␣10␣number");
    for(int i=0;i<10;i++)
    {
        scanf("%d",&s[i]);
    }
    QuickSort(s,0,9);
    for (int j=0;j<10;j++)
    {
        printf("%d␣",s[j]);
    }
}
```

C语言编写的快速排序程序QS

我们只考虑变量程序元素。

将这个代码分成三个模块：令$M = \{m_1 =< E_{m_1}, R_{m_1} >, m_2 =< E_{m_2}, R_{m_2} >, m_3 =< E_{m_3}, R_{m_3} >\}$，其中

- E_{m_1}由第2-18行中语句组成的集合，$E_{m_1} = \{R, i, j, pivot\}$;
- E_{m_2} 由第19-28行中语句组成的集合，$E_{m_2} = \{R, low, high, pivotpos\}$;
- E_{m_3} 由第29-42行中语句组成的集合,$E_{m_3} = \{s[100], i, j\}$。
- $\#(E_{QS}) = 11$（？）。

Claim

变量$i, j \in E_{m_1}$不止一个用途，同样标识符在$main$函数也被使用，所以它们不是用户所期望的。此外，$R \in E_{m_1}$ 和$R \in E_{m_2}$ 不具有自描述性，所以也不是用户所期望的。这样可以得到$\#(E_{m_1})_{expect} = 1$，$\#(E_{m_2})_{expect} = 3$，$\#(E_{m_3})_{expect} = 1$，$\#(E_{QS}) = 11$。从而有$\kappa(m_1) = \frac{1}{11}$，$\kappa(m_2) = \frac{3}{11}$，$\kappa(m_3) = \frac{1}{11}$。

6.5 面向源代码的软件可信性度量模型

Definition (模块间二元经验关系$\succsim_T$)

设$m_1 = (E_{m_1}, R_{m_1})$，$m_2 = (E_{m_2}, R_{m_2}) \in MS$，则模块$m_1$可信性大于等于模块$m_2$可信性的经验关系，记作$m_1 \succsim_T m_2$，当且仅当$\kappa(m_1) \geq \kappa(m_2)$。

Definition (软件模块可信性经验关系系统)

给定一个程序系统$S = (E, R)$，称$(MS, \succsim_T, \circ_T)$为一个软件模块可信性经验关系系统。

6.5 面向源代码的软件可信性度量模型

Theorem

给定一个程序系统$S=(E, R)$，令MS为该系统中所有模块组成的集合，$\Re^+$为正实数集，$m=(E_m, R_m)\in MS$。则函数κ是模块可信性经验关系系统$(MS, \succsim_T, \circ_T)$的实值测量函数，即保持二元闭运算$\circ_T$和二元经验关系$\succsim_T$。

由Extensive结构表示和唯一性定理得到下面结论

Theorem

软件模块可信性经验关系系统$(MS, \succsim_T, \circ_T)$具有*Extensive*结构。

Extensive结构表示定理保证存在从Extensive结构到某数值关系系统具有比率标度的可加函数，并且反之亦然。下面就依据已建立的模块可信性经验关系系统$(MS, \succsim_T, \circ_T)$建立软件模块可信性度量模型。

软件模块可信性经验关系系统的Extensive结构性

Proof.

按照Extensive 结构的定义，我们需要验证4条基本性质：弱序性、弱结合律、单调性和阿基米德性。下面我们逐条进行验证。
首先定义关系$\approx$:
给定两个模块m_1和m_2，$m_1 \approx m_2$ 当且仅当$m_1 \succsim_T m_2$ 与$m_2 \succsim_T m_1$都成立。由关系$\succsim_T$定义可得：$m_1 \approx m_2$当且仅当$\kappa(m_1) = \kappa(m_2)$。
(1) $\succsim_T$是弱序的：验证$\succsim_T$是传递的和强完备的。
(a) 传递性:
(b) 强完备性：
(2) 弱结合律：因为$m_1 \circ_T m_2 = (E_{m_1} \uplus E_{m_2},\ R_{m_1} \cup R_{m_2} \cup R_{m_1 m_2})$，所以

$$(m_1 \circ_T m_2) \circ_T m_3 = ((E_{m_1} \uplus E_{m_2}) \uplus E_{m_2},$$
$$R_{m_1} \cup R_{m_2} \cup R_{m_3} \cup R_{m_1 m_2} \cup R_{m_1 m_3} \cup R_{m_2 m_3} \cup R_{m_1 m_2 m_3})$$

软件模块可信性经验关系系统的Extensive结构性

Proof.

同理，因为$m_2 \circ_T m_3 = (E_{m_2} \uplus E_{m_3},\ R_{m_2} \cup R_{m_3} \cup R_{m_2m_3})$ 所以

$$m_1 \circ_T (m_2 \circ_T m_3) = (E_{m_1} \uplus (E_{m_2} \uplus E_{m_2}),$$
$$R_{m_1} \cup R_{m_2} \cup R_{m_3} \cup R_{m_1m_2} \cup R_{m_1m_3} \cup R_{m_2m_3} \cup R_{m_1m_2m_3})$$

由$\uplus$定义可知$\uplus$满足结合律，且$\cup$满足结合律，所以有

$$(m_1 \circ_T m_2) \circ_T m_3 \approx m_1 \circ_T (m_2 \circ_T m_3)$$

成立。

(3) 单调性：首先证明$\forall m_1$，m_2，$m_3 \in MS$，若$m_1 \succsim_T m_2$，则$m_1 \circ_T m_3 \succsim_T m_2 \circ_T m_3$。

由$m_1 \succsim_T m_2$得$\kappa(m_1) \geq \kappa(m_2)$，所以有

$$\kappa(m_1 \circ_T m_3) = \kappa(m_1) + \kappa(m_3) \geq \kappa(m_2) + \kappa(m_3) = \kappa(m_2 \circ_T m_3)$$

软件模块可信性经验关系系统的Extensive结构性

Proof.

这样有$m_1 \circ_T m_3 \succsim_T m_2 \circ_T m_3$。
接下来证明$\forall m_1$，m_2，$m_3 \in MS$，若$m_1 \circ_T m_3 \succsim_T m_2 \circ_T m_3$，则$m_1 \succsim_T m_2$。
(4) 阿基米德性：任取MS中的模块m_1，m_2，m_3，m_4，再设$m_1 \succsim_T m_2$且$m_2 \not\succsim_T m_1$，则$\kappa(m_1) \geq \kappa(m_2)$且$\kappa(m_1) \neq \kappa(m_2)$。
取正整数n使得$n \geq \dfrac{\kappa(m_4) - \kappa(m_3)}{\kappa(m_1) - \kappa(m_2)}$成立。进而有

$$n\kappa(m_1) + \kappa(m_3) \geq n\kappa(m_2) + \kappa(m_4)$$

成立。这样有

$$nm_1 \circ_T m_3 \succsim_T nm_2 \circ_T m_4$$

成立。
综上所述$(MS, \succsim_T, \circ_T)$具有Extensive结构。 □

作业一

详细证明：软件模块可信性经验关系系统$(MS, \succsim_T, \circ_T)$具有Extensive结构。

6.5 面向源代码的软件可信性度量模型

对于给定一个程序系统$S=(E,\ R)$，令$M(\subseteq MS)$满足

对任意的$e\in E$存在模块 $m=(E_m,\ R_m)\in M$使得 $e\in E_m$且

对任意的$m_i=(E_{m_i},\ R_{m_i})$，$m_j=(E_{m_j},\ R_{m_j})\in M$有$E_{m_i}\cap E_{m_j}=\emptyset$。

这样M是E的一个划分，即由S中互不相交模块组成的集合，并且这些模块包含S当中所有元素。假设M中有n个元素，即S中n个模块，依次表为$m_1,\cdots,m_n$。

Definition (模块权重函数β)

模块权重函数β是从模块集合M到开区间$[0,1]$的一个映射：

$$\begin{array}{rccc}\beta: & M & \to & (0,1)\\ & m_i & \mapsto & \beta(m_i)\end{array}$$

且满足$\sum_{i=1}^{n}\beta(m_i)=1$。

模块权重

- 方法一：按照模块的重要性，构件正互反判断矩阵$(a_{ij})_{n\times n}$，其中$a_{ij}=$模块m_i比模块m_j重要倍数，见第四章；
- 方法二：按照模块所含代码行数的比例—

$$\beta(m)=\frac{\text{模块}m\text{所含有的代码函数}}{\text{代码中参与划分模块的总行数}}$$

C语言编写的快速排序程序QS

```c
#include <stdio.h>
int Partition(int *R,int i,int j)
{
    int pivot=R[i];
    while(i<j)
    {
        while(i<j&&R[j]>=pivot)
            j--;
        if(i<j)
            R[i++]=R[j];
        while(i<j&&R[i]<=pivot)
            i++;
        if(i<j)
            R[j--]=R[i];
    }
    R[i]=pivot;
    return i;
}
void QuickSort(int *R,int low,int high)
{
    int pivotpos;
    if(low<high)
    {
        pivotpos =Partition(R,low,high);
        QuickSort(R,low,pivotpos-1);
        QuickSort(R,pivotpos+1,high);
    }
}//QuickSort
int main()
{
    int s[100];
    printf("input␣10␣number");
    for(int i=0;i<10;i++)
    {
        scanf("%d",&s[i]);
    }
    QuickSort(s,0,9);
    for (int j=0;j<10;j++)
    {
        printf("%d␣",s[j]);
    }
}
```

C语言编写的快速排序程序QS

这个软件含有42行代码，第一行是头文件，后面各模块都要使用它，因而不记载模块中。程序QS可计代码行数为41。
将这个代码分成三个模块：令$M=\{m_1=<E_{m_1}, R_{m_1}>, m_2=<E_{m_2}, R_{m_2}>, m_3=<E_{m_3}, R_{m_3}>\}$，其中

- E_{m_1}由第2-18行中语句组成的集合，含有17行代码，m_1的权重$\beta(m_1)=\frac{17}{41}=0.415$;
- E_{m_2} 由第19-28行中语句组成的集合，含有10行代码，m_2的权重$\beta(m_2)=\frac{10}{41}=0.244$;
- E_{m_3} 由第29-42行中语句组成的集合,含有14行代码，m_3的权重$\beta(m_3)=\frac{14}{41}=0.341$。

6.5 面向源代码的软件可信性度量模型

Definition (模块可信性经验关系系统实值非可加测量函数μ_{na})

给定一个程序系统$S=(E, R)$，令MS为该系统中所有模块组成的集合，$\Re^{+}$为正实数组集，$m=(E_m, R_m) \in MS$，$0<\rho<1$。则模块可信性经验关系系统$(MS, \succsim_T, \circ_T)$的实值测量函数$\mu_2$定义如下：

$$\mu_{na}(m)=(\kappa(m))^{\rho}$$

Remark

由于$0 \leq \kappa(m) \leq 1$，$0<\rho<1$，所以测量函数μ_{na} 是放大了$\kappa(m)$的值，即放大了观察到模块m中具有用户期望机制实现程序元素个数与软件系统E中所有程序元素个数的比值，因此$\mu_{na}(m)$定义了模块m的可信程度。

6.5 面向源代码的软件可信性度量模型

Theorem

μ_{na} 是模块可信性经验关系系统$(MS,\ \succsim_T,\ \circ_T)$的非可加但具有比率刻度度量函数。

证明：令$m_1 = (E_{m_1},\ R_{m_1})$，$m_2 = (E_{m_2},\ R_{m_2})$，$m_3 = (E_{m_3},\ R_{m_3}) \in MS$。因为

$$\mu_{na}(m) = (\kappa(m))^{\rho}$$

所以

$$\kappa(m) = (\mu_{na}(m))^{\frac{1}{\rho}}$$

并且

$$\kappa(m_1 \circ_T m_2) = (\mu_{na}(m_1 \circ_T m_2))^{\frac{1}{\rho}}$$

由命题的结论可得

$$\begin{aligned}\kappa(m_1 \circ_T m_2) &= \kappa(m_1) + \kappa(m_2) \\ &= (\mu_{na}(m_1))^{\frac{1}{\rho}} + (\mu_{na}(m_2))^{\frac{1}{\rho}}\end{aligned}$$

从而有

$$\mu_{na}(m_1 \circ_T m_2) = ((\mu_{na}(m_1))^{\frac{1}{\rho}} + (\mu_{na}(m_2))^{\frac{1}{\rho}})^{\rho}$$

所以μ_{na} 是一个非可加度量函数，即不将模块的二元运算$\circ_T$转换成实数的加法运算$+$。
又因为对于任何正实数γ都有

$$\begin{aligned}\mu_{na}(\gamma(m_1 \circ_T m_2)) &= \left((\gamma\mu_{na}(m_1))^{\frac{1}{\rho}} + (\gamma\mu_{na}(m_2))^{\frac{1}{\rho}}\right)^{\rho} \\ &= \gamma\left((\mu_{na}(m_1))^{\frac{1}{\rho}} + (\mu_{na}(m_2))^{\frac{1}{\rho}}\right)^{\rho} \\ &= \gamma\mu_{na}(m_1 \circ_T m_2)\end{aligned}$$

所以度量模型μ_{na} 具有比率类型。
综上所述结论得证。

6.5 面向源代码的软件可信性度量模型

给定模块m，通过$\mu_{na}(m)$计算后可得到取值范围为$[0,1]$的模块可信度，同样为和第四章基于属性的软件可信性度量模型度量结果取值范围相一致，通过分段函数将取值范围映射到$[1,10]$，即

$$\mu(m)=\begin{cases}1, & \mu_{na}(m)<1/10\\ 10*\mu_{na}(m), & \text{其它情形}\end{cases}$$

6.5 面向源代码的软件可信性度量模型

Definition (面向模块的非线性加法软件可信性度量模型)

给定一个程序系统$S=(E,R)$，其模块系统集$MS=\{m_1,\ldots,m_N\}$，$\beta(m_i)$为模块权重函数，满足$\sum_{i=1}^{N}\beta(m_i)=1$。
面向模块的非线性加法软件可信性度量模型$T_{na}(S)$ 定义如下

$$T_{na}(S)=(\beta(m_1)(\mu(m_1))^{\rho}+\ldots+\beta(m_N)(\mu(m_N))^{\rho})^{\frac{1}{\rho}}$$

6.5 面向源代码的软件可信性度量模型

- 公理化方法通过定义数值关系系统期望具有的性质来对软件属性的经验理解进行形式化描述，其为我们证明所建立模型的合理性提供了一种理论验证方法。
- 我们曾基于该方法提出基于属性的软件可信性度量模型应该具有的多条性质，这些性质基本上反映可信性度量的特点。
- 在这里我们以这些性质为参考建立面向模块的软件可信性度量模型应该具有的性质集，并基于该性质集对所建立度量模型进行验证。

6.5 面向源代码的软件可信性度量模型:性质

为了使验证过程更具一般性，令

$$T(y_1, y_2, \cdots, y_n) = (\beta_1 y_1^{\rho} + \beta_2 y_2^{\rho} + \cdots + \beta_n y_n^{\rho})^{\frac{1}{\rho}}$$

其中，

1. ρ：与模块间可信性替代相关的参数，满足$0 < \rho < 1$；
2. y_i：第$i(i = 1, \cdots, n)$个模块经过模型μ度量结果，满足$1 \leq y_i$；
3. β_i：第$i(i = 1, \cdots, n)$个模块权重，满足$0 \leq \beta_i \leq 1$，$\sum_{i=1}^{n} \beta_i = 1$。

6.5 面向源代码的软件可信性度量模型:性质验证

$$T(y_1, y_2, \cdots, y_n) = (\beta_1 y_1^{\rho} + \beta_2 y_2^{\rho} + \cdots + \beta_n y_n^{\rho})^{\frac{1}{\rho}}$$

Claim

$1 \leq T \leq 10$。

Claim

T满足单调性。

证明：由T定义可知T_{na}为关于y_i的连续可导函数，对T_{na} 关于每一个y_i $(1 \leq i \leq n)$偏导数为

$$\frac{\partial T}{\partial y_i} = \beta_i(\beta_1 y_1^{\rho} + \beta_2 y_2^{\rho} + \cdots \beta_n y_n^{\rho})^{\frac{1}{\rho}-1} y_i^{\rho-1}$$

因为$1 \leq y_i \leq 10$，$0 \leq \beta_i$，$1 \leq i \leq n$，所以$\frac{\partial T}{\partial y_i} \geq 0$，$1 \leq i \leq n$。

6.5 面向源代码的软件可信性度量模型:性质验证

Claim

T 满足凝聚性。

证明：通过对T 关于y_i $(1 \leq i \leq n)$求二阶导数可得

$$\frac{\partial^2 T}{\partial {y_i}^2} = \beta_i(1-\rho)(\beta_1 y_1^{\rho} + \beta_2 y_2^{\rho} + \cdots \beta_n y_n^{\rho})^{\frac{1}{\rho}-2} y_i^{\rho-2}$$

$$\left(\beta_i y_i^{\rho} - (\beta_1 y_1^{\rho} + \beta_2 y_2^{\rho} + \cdots \beta_n y_n^{\rho})\right)$$

因为$0 < \rho < 1$ 且$0 \leq y_i$，β_i，$1 \leq i \leq n$，所以

$$\frac{\partial^2 T}{\partial {y_i}^2} \leq 0, \quad 1 \leq i \leq n$$

6.5 面向源代码的软件可信性度量模型:性质验证

Claim

T满足灵敏性。

经过计算易得

$$\delta_i = \frac{\partial T}{\partial y_i}\frac{y_i}{T} = \frac{\beta_i y_i^{\rho}}{\beta_1 y_1^{\rho} + \beta_2 y_2^{\rho} + \cdots \beta_n y_n^{\rho}}, \quad 1 \le i \le n$$

因为$1 \le y_i \le 10$，$0 \le \beta_i$，$1 \le i \le n$，所以$\delta_i \ge 0$，$1 \le i \le n$。

6.5 面向源代码的软件可信性度量模型:性质验证

$$T(y_1, y_2, \cdots, y_n) = (\beta_1 y_1^{\rho} + \beta_2 y_2^{\rho} + \cdots + \beta_n y_n^{\rho})^{\frac{1}{\rho}}$$

Claim

T 满足替代性。

计算可得

$$\sigma_{ij} = \frac{1}{1-\rho}, \quad 1 \leq i, \ j \leq n, \ i \neq j$$

因为$0 < \rho < 1$。所以T 满足替代性，且具有难易程度为常值的替代性，这里表明参数ρ是反映模块间的替代性。

Claim

T 满足可期望性。

C语言编写的快速排序程序QS

```c
#include <stdio.h>
int Partition(int *R,int i,int j)
{
    int pivot=R[i];
    while(i<j)
    {
        while(i<j&&R[j]>=pivot)
            j--;
        if(i<j)
            R[i++]=R[j];
        while(i<j&&R[i]<=pivot)
            i++;
        if(i<j)
            R[j--]=R[i];
    }
    R[i]=pivot;
    return i;
}
void QuickSort(int *R,int low,int high)
{
    int pivotpos;
    if(low<high)
    {
        pivotpos =Partition(R,low,high);
        QuickSort(R,low,pivotpos-1);
        QuickSort(R,pivotpos+1,high);
    }
}//QuickSort
int main()
{
    int s[100];
    printf("input␣10␣number");
    for(int i=0;i<10;i++)
    {
        scanf("%d",&s[i]);
    }
    QuickSort(s,0,9);
    for (int j=0;j<10;j++)
    {
        printf("%d␣",s[j]);
    }
}
```

C语言编写的快速排序程序QS

将这个代码分成三个模块：令$M=\{m_1=<E_{m_1}，R_{m_1}>，m_2=<E_{m_2}，R_{m_2}>，m_3=<E_{m_3}，R_{m_3}>\}$，其中

- E_{m_1}由第2-18行中语句组成的集合，$\beta(m_1)=0.415$; $\kappa(m_1)=\frac{1}{11}$;
- E_{m_2} 由第19-28行中语句组成的集合，$\beta(m_1)=0.244$; $\kappa(m_2)=\frac{3}{11}$;
- E_{m_3} 由第29-42行中语句组成的集合,$\beta(m_1)=0.341$; $\kappa(m_3)=\frac{1}{11}$。

C语言编写的快速排序程序QS序

假定$\rho = 0.8$，则得$\mu_{na}(m_1) = (\kappa(m_1))^{\rho} = 0.1^{0.8} = 0.147$,
$\mu_{na}(m_2) = (\kappa(m_2))^{\rho} = 0.27^{0.8} = 0.354$,$\mu_{na}(m_3) = (\kappa(m_1))^{\rho} = 0.1^{0.8} = 0.147$,因此

$$\begin{aligned}
\mu(m_1) &= \mu_{na}(m_1) = 10 * (\mu_{na}(m_1)) = 1.47 \\
\mu(m_2) &= \mu_{na}(m_2) = 10 * (\mu_{na}(m_2)) = 3.54 \\
\mu(m_3) &= \mu_{na}(m_3) = 10 * (\mu_{na}(m_3)) = 1.47
\end{aligned}$$

$$\begin{aligned}
T_{na}(S) =& (\beta(m_1)(\mu(m_1))^{\rho} + \beta(m_2)(\mu(m_2))^{\rho} + \beta(m_3)(\mu(m_3))^{\rho})^{\frac{1}{\rho}} \\
=& (0.415 \times 1.47^{0.8} + 0.244 \times 3.54^{0.8} + 0.341 \times 1.47^{0.8})^{\frac{1}{0.8}} \\
=& 1.94
\end{aligned}$$

C语言编写的快速排序程序QS

影响非可加可信性度量模型可信性度量值的参数共有两类：ρ和β。权重函数β有了比较合理性的构建方法。由于软件模块间的可信性替代性度$\delta_{ij}=\frac{1}{1-\rho}$，很明显当$\rho$减少时，模块间的替代性也在减少，但软件可信度却在增加。

ρ	T_{na}	δ_{ij}
0.8	1.94	5
0.6	2.86	2.5
0.4	4.30	1.67
0.2	6.54	1.25
0.1	8.08	1.11
0.01	9.79	1.01

Remark

从这个例子可以看到ρ取小于0.1的值比较合适。

作业二

- 编程实现面向源代码的软件可信度量模型T_{na};
- 将kruskal算法程序分成8个模块$m_1:9-17$行，$m_2:18-39$行，$m_3:40-48$行，$m_4:49-53$行，$m_5:54-60$行，$m_6:61-70$行，$m_7:71-87$行，$m_8:88-114$行。模块权重函数β使用模块所含有的代码行数量计算。参数$\rho=0.8,0.6,0.4,0.2,0.1,0.01,0.001$计算软件的可信度。度量$\kappa$是模块中用户期望机制实现变量元素个数与模块中所有变量个数的比值。

6.6 面向源代码的软件可信性规范

6.6 面向源代码的软件可信性规范

- 在前一节，我们给个简单例子来说明如何通过模块来计算软件的可信度。在计算过程中，除了两个计算模型外，还统计了具有用户期望机制实现程序元素个数，突出了用户期望。
- 本节借鉴Dromey构建软件质量模型的思想，为每一种程序元素和程序单元构造可信规范。为此构建面向源代码的可信性规范，作为用户关于程序元素的期望实现机制。
- 在给出为程序元素和程序单元构造的可信规范之前，有两点必须做一说明。
  - 首先，我们所给出的规范不具有完备性，用户可以根据需要对规范进行扩充，也可以根据需要只选择规范中的一部分来使用;
  - 其次，每一条可信规范对软件可信性影响不一定相同，我们虽然考虑到这种情形，但为了简单起见，并没有对其做深入探讨。

6.6 面向源代码的软件可信性规范

Table: 变量可信规范

所有变量是否都已经声明？
如果一个变量的某些属性在声明中没有明确给出，则其默认情形是否可以很好地理解？
变量是否有声明为有固定的上下界？
所有声明的变量是否都已经初始化？如果初始化，是否正确地初始化？
所有声明的变量是否都用到了？
每个变量是否都有且只有一个用途？
每个变量名是否有自描述性？
同一变量是否没有类似的名字？
当布尔变量的值为真时，变量名能否准确表达其含义？
如果存在全局变量，在所有引用全局变量的模块中，全局变量的定义和属性是否相同？
一个变量的类型在所有使用地方是否都一致？
类型转换是否显式进行？

6.6 面向源代码的软件可信性规范

Table: 常量可信规范

代码中是否避免了”奇异数”（常数）？
如果存在数学常量是否只有一种表示？（例如，不是在一个地方用3.14159，而在另一个地方用3.1416）
是否一致地使用了命名常量，而不是一处使用命名常量，另一处使用数值？
代码中是否避免了常数型字符和字符串？
对字符串引用是否避免了边界错误？

6.6 面向源代码的软件可信性规范

Table: 数组可信规范

数组长度是否非固定？
数组所有元素的功能是否相关？
数组下标的类型是否为整型？
数组下标在使用之前是否检查其范围？是否对所有下标的引用都没有发生越界错误？
多维数组下标排列顺序正确吗？
是否使用大括号等类似界符以指示和匹配数组的初始化构造？

6.6 面向源代码的软件可信性规范

Table: 算术表达式可信规范

所有功能类似的表达式是否都用类似的方法进行构造？
是否有括号或者其它技术用来在算术表达式的计算中消除模糊？
是否避免数量级相差过大的数之间加减运算？
程序中是否系统地采取措施来防止舍入误差问题？
在计算表达式中是否有变量具有不一致类型？如一个字符型变量和一个整型变量相加或者一个浮点类型变量和一个整数类型变量相加。如果有，是否对表达式按照你的意愿进行求值？
在除法运算中除数是否有可能为0？
是否对负数进行开平方运算？
是否有计算用具有相同数据类型但不同长度的变量？如一个整数型2维数组和一个整数型1维数组相加。
计算过程中是否会出现上溢或者下溢的情形？尤其是关于整数运算，即最终计算结果看起来似乎是正确的，但中间计算结果相对于程序语言的数据类型而言太大或太小。
对于包含多个运算符的表达式，关于运算符的优先级假设是否正确？
关于整数类型的算术表达式使用是否确？尤其是包含有除法运算的整数类型算术表达式？$inti2 * i/2 == i$ 上式表达式是否成立依赖于i为奇数还是偶数，并依赖于乘法运算与除法运算的优先级。
是否存在计算冗余？

6.6 面向源代码的软件可信性规范

Table: 逻辑表达式可信规范

是否存在不同类型变量的比较？比如字符型变量和数值型变量之间的比较？
是否存在混合模式比较或者不同长度变量的比较？如果存在，是否确保很好地理解了转换规则？
程序中是否避免了对浮点数进行相等比较？
是否不存在结果会变的逻辑运算？
比较运算符是否正确？程序员经常对至多、至少、大于、不少于、少于或者等于混淆。
逻辑运算符的运算对象是否正确？比较和逻辑运算符是否完全分开明确？
在包含多个逻辑运算符的逻辑表达式中，计算顺序和运算符优先级的假设是否正确？
编译器对逻辑运算符的计算方式是否不会影响程序结果？
是否不存在逻辑冗余？

6.6 面向源代码的软件可信性规范

Table: 赋值语句可信规范

一个变量是否赋予正确的长度和数据类型？

Table: 顺序语句可信规范

语句间的依赖关系表示是否清楚吗？若代码的依赖关系不清楚，用注释注明了吗？
是否把相关语句组织在一起？

Table: 条件语句可信规范

IF-THEN 语句在出现等号时流向是否正确？
ELSE 语句是否存在？
全部情况是否都覆盖了？

6.6 面向源代码的软件可信性规范

Table: 循环语句可信规范

每个循环是否都会终止？是否可以设计一个非形式化的证明或者讨论说明每一个循环都会终止？
循环开始之前是否进行初始化？如果是，初始化是否靠着循环顶部进行？
无限循环是否没有使用$while(true)$表示？是否使用诸如$for\ i := 1\ to\ 9999$之类更清楚地表示吗？
是否循环体用 或类似的结构去标明以免在修改时出错？
在嵌套循环中，作为循环变量的数组下标是否正确？是否不存在出现交叉错误？
循环是否完成一个且仅完成一个功能吗？
循环下标的名字是否有意义吗（如果循环的长度超出了一两行代码或者出现了嵌套循环，那么就应该是i、j或者k以外的其它名字）？
循环是否在所有可能情况下都能退出吗？
循环的中止条件是否明显？
循环控制变量是否是整数类型？
循环控制变量参数在使用前是否检查其范围？
循环控制变量是否仅有一个有含义的名字？

6.6 面向源代码的软件可信性规范

Table: 程序单元可信规范

程序单元是否包含注释块？用来描述程序名、准确需求、目的、限制、修改历史、输入输出、方法、假设、对所有可预见错误的错误恢复程序，和它进行通讯的模块信息等。
所有程序单元的输入输出是否都被识别和描述？
程序单元是否声明了？
在程序单元的声明中是否为所有参数给出了标识符？
程序单元的声明和定义中使用的标识符是否一致？
程序单元是否定义为不带有可变数量的参数？
所有使用程序单元的地方和其定义的类型是否一致？
每个程序单元是否都检查了其输入参数正确性？
是否用到了每一个输入参数？
是否不存在不完整输出或者缺失输出问题？
是否用到了每一个输出参数？
带有non-void 返回类型函数的所有退出路径是否有显式带表达式的return 语句？
实际参数与形式参数是否匹配？包括参数的数目、顺序、类型和单位等。
常数是否没有作为形参？
是否避免了程序单元中的参数用在工作变量？
重复的代码是否封装在一个程序单元中？
当错误发生时，程序是否有能力展现清楚有用的出错信息？

6.6 面向源代码的软件可信性规范

Table: 程序单元可信规范(续)

程序片段是否明确地分离？
是否按照顶端注释块、规范语句，然后可执行代码顺序写程序？
每一行代码是否至多有一个可执行语句？
在程序中是否用相同的缩进方式
程序是否被构造成功能独立的模块？
存在单元依赖的地方，是否通过注释、程序文档或内在程序结果明确说明？
程序是否能按要求显示中间结果？
程序是否包含提供测试用例的识别或者对输出测试结果的描述？
程序单元是否包含检查点重启动的能力？
所有程序单元是否都包含至多一个输出点？
程序是否允许对没致命错误的恢复？
当有错误事件发生时，程序是否提供诊断信息显示？
程序控制是否通过输入可以明确地指明？
程序的功能、输入、输出是否都被充分地定义方便对程序进行测试？
注释是否提供支持特殊输入值的选择来允许专门程序性能测试？
对于选择的操作模式，程序是否可以省略提供不必要的输入、计算和输出？

6.7 本章小结

- 本章将物理科学中度量理论应用于软件可信性度量中，构建关于软件可信性具有Extensive结构的经验关系系统，并建立基于该结构的非线性加法度量模型。
- 给出基于模块的软件可信性度量模型应该满足的性质集，利用公理化方法对所建立的非线性加法度量模型进行验证，从而最终建立同时经过度量理论和公理化方法验证的面向源代码的软件可信性度量模型。
- 将来工作主要包括以下方面：完善关于程序元素和程序单元的可信规范，对所建立模型进行实验验证；构建基于物理科学中其它度量结构的软件可信性度量模型。