论程序设计课程教学中的同构化创新思想教育*)

——“对→好→巧→妙→绝”的算法设计案例

周启海　李朔枫　杨祥茂　黄　涛

（西南财经大学经济信息工程学院　成都 610074）

摘　要　本文基于同构化基本原理，提出了“算法是程序之母，程序是算法之子”的程序设计教学指导原则；并通过“对→好→巧→妙→绝”的典型同构化算法设计创新案例，阐明了应该和可以“寓同构化创新思想教育于算法先导型程序设计课程教学中”的新理念、新方法与新实践。

关键词　同构化，算法先导，程序设计，创新教育

On the Education for the Thinking of Isomorphic Creativity in Teaching of Programming Courses

——A Case Study of Algorithm Design with the “Right →Good →Clever →Excellent →Wonderful”

ZHOU Qi-Hai　LI Shuo-Feng　YANG Xian-Mao　HUANG Tao

(School of Economic Information Engineering, Southwestern University of Finance and Economics, Chengdu 610074)

Abstract　In this paper, based on the isomorphic principles, the guiding principle of programming described as “an algorithm is the mother of its programs, and the programs are the sons of their algorithm” is raised; by means of some typical creative cases of an isomorphic algorithm design with the “Right→Good→Clever→Excellent→Wonderful”, a new idea, new way and new practice of “the education for the thinking of isomorphic creativity which should and could exist in teaching of programming courses based on the Pre-guide of algorithm design” are clarified.

Keywords　Isomorphic, Pre-guide of algorithm, Programming, Creativity education

“探求真知，追求创新”，是现代教育的精髓。当今程序设计课程教学中，应提倡和鼓励“创新理念→创新思维→创新实践”的创新教育思想。基于周启海教授1986年提出的同构化基本原理[1]，本文仅以“判定任给自然数是否回文数（即顺向、反向读它均为同一个数的自然数）”为例，阐明在程序设计教学中，如何经由“对→好→巧→妙→绝”的算法设计优化之路，来体现同构化创新思想教育。

1　从“对”出发的算法设计

从学习编程（即设计算法、编写程序）伊始，就该懂得“算法，是程序之母；程序，是算法之子”，应努力探索、追求创新：循着“对→好→巧→妙→绝”编程优化之路，去寻求解决同一问题的算法（含其程序）族中的佼佼者——最优（或满意）算法。确保“对”（指算法正确），无疑是算法设计首要前提。请

```
算法  Eg01                    {“取n高位数字另作低位数字”的回文数判定算法1。它只算“对”！}
 {{整型：i,k,m,n,p,s}};       {整型变量定义}
>>>                           {算法开始}
输出 "输入待判定自然数  n=?";  {（非换行方式）显示输入提示信息}
输入 n;                       {输入待判定的任给自然数}
输出 "自然数",n;              {（非换行方式）回声输出所给的待判定自然数n}
m←n;                          {用m标记所输入的自然数n}
k←0;                          {“统计自然数m(即自然数n)位数”的计数器初始化}
 @: \\                         {（直到型循环结构的）循环体由此开始}
   m←m DIV 10;                {用“求m除以10的商数”，来求“去掉m的个位数后所剩之自然数”}
  k←k+1                       {统计自然数m的位数}
 //                           {（直到型循环结构的）循环体到此结束}
直到 m=0;                     {直到型循环结构的循环条件判定：直到自然数m位数统计完毕}
k←k-1;                        {求得自然数n的初始最高权重10^k之指数k}
m←n; s←0;                     {标记自然数n；并使“生成自然数n的反向数”累加器s取初值}
对 i←0,k @: \\                 {（步长型循环结构，其初值0、终值k、步长1）循环体由此开始}
  p←10**(k-i);                {求当前自然数m的最高权重。其中：10**(k-i)表示10^(k-i)}
  s←s+INT(m/p)*(10**i);       {累加生成自然数n的反向数。其中：INT(m/p)表示对m/p取整}
  m←m MOD p                   {求m除以p的余数，即“去掉m的最高位数后所剩之自然数”}
 //;                           {（步长型循环结构）循环体到此结束}
如果 s=n                      {自然数n与其反向数s相同吗？}
  T: 行输出 "是回文数！"      {（“T:”表示“为真”，即）成立，就输出“是回文数！”的信息}
  F: 行输出 "非回文数！";     {（“F:”表示“为假”，即）不成立，就输出“非回文数！”的信息}
!!!                           {算法结束}
```

先看一个虽然正确，但只能算“对”的算法Eg01。

---

*)**周启海**　教授，博（硕）士生导师，主要研究方向：计算几何，算法研究与实现，财经计算，同构化信息处理等。**李朔枫**　副教授，硕士生导师，主要研究方向：计算机应用，电子商务。**杨祥茂**　副教授，硕士生导师，主要研究方向：计算机应用。**黄　涛**　讲师，主要研究方向：计算机应用。

2 为"好"前行的算法设计

设计"对"的算法，只是"万里长征"的第一步，应为设计更"好"的算法而继续前行。例如：算法 Eg01 只能算"对"，因为它受其瓶颈"必须首先判断出任给自然数 n 的位数，才能实现对 n 取高位数字作为低位数字"的制约，使运行时既浪费过多时间又占用过大空间。为此，必须多角度、多方位思考问题，以提高时间效率与空间效率。事实上，只要灵活运用"对自然数 n 取常用对数"的方法，就可直接快速确定出 n 的位数。据此，算法 Eg02A 较好地解决了困扰算法 Eg01 的"位数判定"瓶颈，使运算速度大大提高——既减少了运算时间，又节约了空间，成为时空效率同时优化的"好"算法。

```
算法 Eg02A                    {"对 n 取常用对数" 的回文数判定算法 2。它算是 "好"! }
{{整型: i,k,m,n,p,p1,s}};     {整型变量定义}
>>>                           {算法开始}
输出 "输入待判定自然数  n=?"; 输入 n; 输出 "自然数",n;{提示下输入待判定的任给自然数}
m←n; k←INT(lg(n))+1;          {直接确定 n 为 k 位数。"lg(n)" 表示 "取 n 的常用对数" }
p←10**k;                      {生成 n 的初始最高权重 p}
s←0; p1←1;                    {自然数 n 的反向数累加器 s、权重累乘器 p1 初始化}
对 i←1,k @: \\                {m 的位数控制}
  p←p DIV 10;                 {生成 m 的当前最高权重 p}
  s←s+(m DIV p)*p1;           {累加生成自然数 n 的反向数}
  m←m MOD p;                  {求 m 除以 p 的余数，即 "去掉 m 的最高位数后所剩数" }
  p1←p1*10;                   {生成 n 的反向数 s 的下一最高权重 p1}
 //;                           {循环体到此结束}
如果 s=n                      {n 的反向数 s 就是 n 吗? }
  T: 行输出 "是回文数! "      {(换行方式) 输出 "是回文数! " 的信息}
  F: 行输出 "非回文数! "      {(换行方式) 输出 "非回文数! " 的信息}
!!!                           {算法结束}
```

又如：算法 Eg02B —改算法 Eg01、算法 Eg02A"对 n 取高位数字作为低位数字"的朴素思想，而直接采用"对 n 取低位数字作为高位数字"的逆向转换方法，从而免去确定 n 的位数，疏通了算法 Eg01 的"位数判定"瓶颈，成为殊途同归的"好"算法。

```
算法 Eg02B                    {"取低位作为高位" 的回文数判定算法 3。它算是 "好"! }
{{整型: i,k,k1,n,n1,n2,  p,p1,s}}; {整型变量定义}
>>>                           {算法开始}
输出 "输入待判定自然数  n=?"; 输入 n; 输出 "自然数",n;{提示下输入待判定的任给自然数}
m←n; s←0;                     {标记自然数 n; 初始化 "取低位作为高位" 累加器 s}
当 m<>0 @: \\                 {当 m 的各个数字尚未取完时}
  s←s*10+m MOD 10;            {"取低位作为高位" 累加生成自然数 n 的反向数 s}
  m←m DIV 10                  {去掉 m 的已处理的个位数字}
  //;                          {当型循环体到此结束}
如果 n=s                      {自然数 n 与其反向数 s 相同吗? }
  T: 行输出 "是回文数! "      {输出 "是回文数! " 的信息}
  F: 行输出 "非回文数! ";     {输出 "非回文数! " 的信息}
!!!                           {算法结束}
```

3 往"巧"前进的算法设计

有了"好"的算法，还不能自我满足、裹足不前，而应继续提高算法质量，向"巧"前进。比如：算法 Eg02A、Eg02B 虽较"好"，但只需细看，就会发现它们都还存在不足：试想，若所给自然数的位数十分长，则用算法 Eg02A、Eg02B 还是要耗费较多时间去处理相当多的每一位数。若利用回文数的对称性，就只需处理自然数 n 的前一半所得数，将进一步节约时间。于是，可得更"巧"算法 Eg03A。

```
算法 Eg03A                    {取对数、取前半的回文数判定算法 4。它已是 "巧"! }
{{整型: i,k,k1,n,n1,n2,p,p1,s}}; {整型变量定义}
>>>                           {算法开始}
输出 "输入待判定自然数  n=?"; 输入 n; 输出 "自然数",n;{提示下输入待判定的任给自然数}
k←INT(lg(n))+1; k1←k DIV 2;   {直接确定 n 为 k 位数; 并取得 n 的位数 k 之半 k1}
p←10**k1;                     {生成把 n 分为前、后两半所得数的基准权重 p}
n1←n DIV p; n2←n MOD p;       {把 n 分为前一半、后一半所成数 n1、n2}
如果 k MOD 2=1                {n 的位数 k 是奇数吗? }
  T: n1←n1 DIV 10;            {剔除位于对称中心处的 n 之中点数字}
s←0; p1←1;                    {自然数 n1 的反向数累加器 s、权重累乘器 p1 初始化}
对 i←1,k1 @: \\               {(n 的前一半所成数 n1) 位数控制}
  p←p DIV 10;                 {生成 n1 的当前最高权重 p}
  s←s+ (n1 DIV p) *p1;        {累加生成自然数 n1 的反向数}
  n1←n1 MOD p;                {求 n1 除以 p 的余数，即 "去掉 n1 的最高位数后所剩数" }
  p1←p1*10;                   {生成 n1 的反向数 s 的下一最高权重 p1}
 //;                           {循环体到此结束}
如果 s=n2                     {n 的前一半数 n1 的反向数 s 就是 n 的后一半数 n2 吗? }
  T: 行输出 "是回文数! "      {输出 "是回文数! " 的信息}
  F: 行输出 "非回文数! "      {输出 "非回文数! " 的信息}
!!!                           {算法结束}
```

同理，算法 Eg02B 也只需处理 n 的后一半所得数，即得会令读者豁然开朗的更"巧"的算法 Eg03B。

```
算法 Eg03B                    {低作高、取后半的回文数判定算法5。它已是“巧”！}
{{整型：k,n,n1,n2, s}};       {整型变量定义}
>>>                           {算法开始}
输出 "输入待判定自然数  n=?"; 输入 n; 输出 "自然数",n;{提示下输入待判定的任给自然数}
k←INT(lg(n))+1;               {直接确定 n 为 k 位数}
p←10**(k DIV 2);              {生成把 n 分为前、后两半所得数的基准权重 p}
n1←n DIV p; n2←n MOD p;       {把 n 分为前一半、后一半所成数 n1、n2}
如果 k MOD 2=1                {n 的位数 k 是奇数吗？}
  T: n1←n1 DIV 10;            {剔除位于对称中心处的 n 之中点数字}
s←0;                          {初始化“取低位作为高位”累加器 s}
当 n2<>o @: \\                {当 n 的后一半数 n2 的各个数字尚未取完时}
  s←s*10+n2 MOD 10;           {“取低位作为高位”累加生成自然数 n2 的反向数 s}
  n2←n2 DIV 10                {去掉 n2 的已处理的个位数字}
 //;                           {循环体到此结束}
如果 n1=s                     {n 的前一半数 n1 就是 n 的后一半数 n2 的反向数 s 吗？}
  T: 行输出 "是回文数！"      {输出“是回文数！”的信息}
  F: 行输出 "非回文数！";     {输出“非回文数！”的信息}
  !!!                         {算法结束}
```

4 朝“妙”迈进的算法设计

在达到了“巧”之后，就应向“妙”迈进：即在“巧”的算法基础上，找出可望改进之处，以进一步优化算法设计。譬如：经过深入思考，可发现“巧”算法 Eg03A、Eg03B 所采用的“把自然数 n 分为前、后两半部分，并只转换某一半部分，再与另一半作比较”，其实质是判断后半部分的低位与前半部分的高位数字是否对应相等。据此，可将 n 的前、后半部分的数字分别从其高位、低位起，同时进行“对称分离、头尾比较”，一旦发现不等，即能判定此数不是回文数（特别是对于较长的数据，它将节省不少时间）。由此所产生的新颖算法，可谓“妙哉”。

同理，算法 Eg03B 也只需将 n 的前、后半部分的数字分别从其高位、低位起，同时进行“对称分离、头尾比较”处理，即可得比它更“巧”的算法 Eg04B。

```
算法 Eg04A                    {对称分离、头尾比较、前半为主的回文数判定算法6。它已是“妙”！}
{{整型：i,k,k1,n,n1,n2,p,p1,p2,s}}; {整型变量定义}
>>>                           {算法开始}
输出 "输入待判定自然数  n=?"; 输入 n; 输出 "自然数",n;{提示下输入待判定的任给自然数}
k←INT(lg(n))+1; k1←k DIV 2;   {直接确定 n 为 k 位数；并取得 n 的位数 k 之半 k1}
p←10**k1;                     {生成把 n 分为前、后两半所得数的基准权重 p}
n1←n DIV p; n2←n MOD p;       {把 n 分为前一半、后一半所成数 n1、n2}
如果 k MOD 2=1                {n 的位数 k 是奇数吗？}
  T: n1←n1 DIV 10;            {剔除位于对称中心处的 n 之中点数字}
s1←0; p1←1; p2←10;            {n1 反向数累加器 s、权重累乘器 p1、n2 权重累乘器 p2 初值}
i←1;                          {从 n 的前一半所成数 n1 的第 1 位起，进行判定}
当 i<=k1 @: \\                {（n 的前一半所成数 n1）位数控制}
  p←p DIV 10;                 {生成 n1 的当前最高权重 p}
  s←s+ (n1 DIV p) *p1;        {累加生成自然数 n1 的反向数}
  n1←n1 MOD p;                {求 n1 除以 p 的余数，即“去掉 n1 的最高位数后所剩数”}
  p1←p1*10; p2←p2*10;         {生成 n1 的反向数 s 的下一最高权重 p1}
  如果 s<>n2 MOD p2           {前半数 n1 的当前反向数 s 不是后半数 n2 对应数吗？}
    T: i←k1+2                 {设置“一旦出现非回文数，就跳出循环”的强制中止标志}
 //;                           {循环体到此结束}
如果 i<>k1+2                  {n 的前一半数 n1 的反向数 s 就是 n 的后一半数 n2 吗？}
  T: 行输出 "是回文数！"      {输出“是回文数！”的信息}
  F: 行输出 "非回文数！"      {输出“非回文数！”的信息}
  !!!                         {算法结束}
```

```
算法 Eg04B                    {对称分离、头尾比较、后半为主的回文数判定算法7。它已是“妙”！}
{{整型：k,k1,n,n1,n2,p,s}};   {整型变量定义}
>>>                           {算法开始}
输出 "输入待判定自然数  n=?"; 输入 n; 输出 "自然数",n;{提示下输入待判定的任给自然数}
k←INT(lg(n))+1; k1←k DIV 2;   {直接确定 n 为 k 位数；并取得 n 的位数 k 之半 k1}
p←10**k1;                     {生成把 n 分为前、后两半所得数的基准权重 p}
n1←n DIV p; n2←n MOD p;       {把 n 分为前一半、后一半所成数 n1、n2}
如果 k MOD 2=1                {n 的位数 k 是奇数吗？}
  T: n1←n1 DIV 10;            {剔除位于对称中心处的 n 之中点数字}
s←0;                          {n2 反向数累加器 s、权重累乘器 p1、n2 权重累乘器 p2 初值}
i←1;                          {从 n 的前一半所成数 n1 的第 1 位起，进行判定}
当 i<=k1 @: \\                {（n 的前一半所成数 n1）位数控制}
  s←s*10+n2 MOD 10;           {“取低位作为高位”累加生成自然数 n2 的反向数 s}
  n2←n2 DIV 10                {去掉 n2 的已处理的个位数字}
  p←p DIV 10;                 {生成取 n 前半数的对应部分搞位数的当前权重 p}
  如果 s<>n1 MOD p            {前半数 n2 对应部分数不是后半数 n2 的当前反向数 s 吗？}
    T: i←k1+2                 {i 取强制中止标志值：一旦出现非回文数，就中断循环！}
    F: i←i+1                  {准备处理 n 的下一双“头、尾”对应数}
 //;                           {循环体到此结束}
如果 i<>k1+2                  {n 的前一半数 n1 的反向数 s 就是 n 的后一半数 n2 吗？}
  T: 行输出 "是回文数！"      {输出“是回文数！”的信息}
  F: 行输出 "非回文数！";     {输出“非回文数！”的信息}
  !!!                         {算法结束}
```

5 向“绝”挺进的算法设计

走过算法优化“对→好→巧→妙”阶段后，应向其最高境界的“绝”顶挺进。例如，由“巧”到“妙”的算法 Eg04A、算法 Eg04B 的思想已与它前面的各个算法有了本质的区别和根本的提升，因为它不用处理完任给自然数 n 的所有数位，就能判定 n 不是回文数。它是否已臻于完美呢？答案是否定的。只要重新深入审视“头尾比较”这一述算法设计思想，不难看出：任给自然数 n 前半部分的高位数字，就是 n 后半部分的对应低位数字。故不必把 n 从中间断开、一分为二，而可直接对整个 n 进行处理，以省去判断 n 位数的奇偶性之麻烦，使得程序更显简洁、快捷。这一突破性的改进，就得到了如下堪称“绝”的最优算法 Eg05A。

```
算法 Eg05A                    {对称分离、只取数字、头尾比较的回文数判定算法 8。它已是“绝”！}
{{整型：i,k,n,p,s}};          {整型变量定义}
>>>                           {算法开始}
输出 "输入待判定自然数 n=?"; 输入 n; 输出 "自然数",n;{提示下输入待判定的任给自然数}
p←INT(lg(n)); k←(p+1) DIV 2;  {直接确定 n 为 p 位数；并取得 n 的位数 p 之半 k}
p←10**p;                      {生成取 n 当前最高位数字的基准权重 p}
i←1;                          {从 n 的第 1 位起，进行判定}
当 i<=k @: \\                 {（n 的“头、尾”对应数字）对数控制}
  如果 n DIV p<>n MOD 10      {n 的当前“头、尾”（即最高、最低位）数字不相同吗？}
    T: i←k+2                  {i 取强制中止标志值：一旦出现非回文数，就准备中断循环！}
    F: \\
      n←(n MOD p) DIV 10;     {去掉已处理的 n 的当前最高位、最低位数字}
      p←p DIV 100;            {生成 n 的当前最高位数字权重 p}
      i←i+1                   {准备处理 n 的下一双“头、尾”对应数字}
   //
 //;                          {循环体到此结束}
如果 i<>k+2                   {n 的所有头尾对应数字均相同吗？}
  T: 行输出 "是回文数！"      {输出“是回文数！”的信息}
  F: 行输出 "非回文数！";     {输出“非回文数！”的信息}
 !!!                          {算法结束}
```

应当指出：解决同一问题的算法设计，往往“仁者见仁，智者见智，各怀高见，均有奇招”。例如：以上各个算法，都是基于“用数值型进行数据处理”的常规思路，走着“对→好→巧→妙→绝”的编程优化之路。然而，如果另辟捷径——“用字符串进行数据处理”的超常思路，也可展示其绝招：在输入待判定自然数时，使逐次输入的其当前位数字，以数字串（其长度为 1）的形式，分别与生成顺向数串、反向数串累加器的 s1、s2 相拼接，就能最后得到所给自然数的顺向数串、反向数串 s1、s2。然后，根据“只有顺向数串 s1 与反向数串 s2 相同的自然数才是回文数”，就可同样得使人拍案叫“绝”的算法 Eg05B。

```
算法 Eg05B                    {直取数串、顺反串累加的回文数判定算法 9。它更是“绝”!!}
{{字符型：数组 x[1]}};        {一维字符型数组变量定义}
{{字符串型：s1,s2}};          {字符串型变量定义}
>>>                           {算法开始}
输出 "输入待判定自然数 n=?";{提示用户输入自然数 n}
s1←""; s2←"";                 {字符初值（空串）}
@: \\                         {（直到型循环结构的）循环体由此开始}
  输入 x[1];                  {（每次一个）逐次输入待判定的任给自然数的当前位数字}
  s1←s1+x; s2←x+s2;           {用字符串累加器 s1、s2，分别逐步生成顺向、反向数串}
 // 直到 ASC(x[1])=10;        {直到最后所输入字符的 ASCII 码为（换行符的 ASCII 码）10}
如果 s1<>s2                   {所给自然数对应的顺向、反向数串均相同吗？}
  T: 行输出所给自然数",s1, "是回文数！"    {输出“是回文数！”的信息}
  F: 行输出所给自然数",s1, "非回文数！"    {输出“非回文数！”的信息}
   !!!                        {算法结束}
```

此外，如果用字符型数组各下标变量顺次存放、对称比较所给自然数各位数字，也将有异曲同工之妙：

```
算法 Eg05C                    {用字符型数组存放自然数的回文数判定算法 10。它也是“绝”!!!}
{{字符型：数组 x[1000]}};     {一维字符型数组变量定义}
{{整型：i,k,n}};              {整型变量定义}
>>>                           {算法开始}
输出 "输入待判定自然数 n=?";{提示用户输入自然数 n}
i←0;                          {字符计数器初值}
@; \\                         {（直到型循环结构的）循环体由此开始}
  i←i+1; 输入 x[i]; 输出 x[i]; {（每次一个）逐次输入待判定的任给自然数的当前位数字}
 // 直到 ASC(x[1])=10;        {存放数：直到最后所输入字符的 ASCII 码为（换行符的 ASCII 码）10}
n←i-1; k←n/2; i←1;            {求双端对称位置数字比较所需“对”的个数 k}
当 i<=k @:                    {当双端对称位置数字尚未比较完时}
  如果 a[i]=a[n-i+1]          {双端对称位置的数字相同吗？}
   T: i←i+1                   {准备进行下一对双端对称位置数字的比较}
   F: i←k+2;                  {设置“双端对称位置数字出现不同”标志，兼作强制中止循环标志}
如果 i<k+2                    {“双端对称位置数字没有出现不同相同”吗？}
  T: 行输出所给自然数",s1, "是回文数！"    {输出“是回文数！”的信息}
  F: 行输出所给自然数",s1, "非回文数！"    {输出“非回文数！”的信息}
 !!!                          {算法结束}
```

限于篇幅，本文仅给出令人拍手称“绝”的算法 Eg05C 所对应的 C 语言程序如下：

```
/*C 语言程序 Eg05C.C          用字符型数组存放自然数的回文数判定算法 10。它也是“绝”!!! */
#include <stdio.H>
char a[1000];                 /*一维字符型数组变量定义*/
int i,k,n;                    /*整型变量定义*/
void main() {
  printf("\n 输入待判定自然数 n=");  /*提示用户输入自然数 n*/
  i=0;                        /*字符计数器初值*/
  do { i++; a[i]=getchar(); putchar(a[i]); }
   while (a[i]!=10);          /*存放数：直到最后所输入字符的 ASCII 码为（换行符的 ASCII 码）10*/
  n=i-1; k=n/2; i=1;          /*求双端对称位置数字比较所需“对”的个数 k*/
  while (i<=k)                /*当双端对称位置数字尚未比较完时*/
    if (a[i]==a[n-i+1])       /*双端对称位置的数字相同吗？*/
      i++;                    /*准备进行下一对双端对称位置数字的比较*/
     else i=k+2;              /*设置“双端对称位置数字出现不同”标志，兼作强制中止循环标志*/
  if (i!=k+2)                 /*"双端对称位置数字没有出现不同相同"吗？*/
   printf("是回文数!\n");     /*输出“是回文数！”的信息*/
     else
   printf("非回文数!\n");     /*输出“非回文数！”的信息*/
  }
```

结论 作者长期教学的成功实践证明：在程序设计课程中，应该而且可以通过不断追求“对→好→巧→妙→绝”的同构化创新教育，来逐步培养和训练学生在校期学习与毕业后工作中，养成可受益终身的“积极观察、认真思考，努力探索、力求创新”的创新意识、新理念、创新思索与创新实践。

参 考 文 献

1 周启海. C++同构化对象程序设计原理[M]. 北京：清华大学出版社，北方交通大学出版社，2004

2 周启海. C 语言程序设计[M]. 北京：机械出版社，2004

3 周启海. C 语言程序设计新捷径[M]. 上海：复旦大学出版社，2000

4 吴红玉，周启海，杨祥茂. 论 JAVA 程序设计教学中的同构化创新思想教育. 见：程序设计语言及其教学探索. 北京：清华大学出版社，2006. 165～168

5 张元新，周启海，杨祥茂. 论 C 程序设计教学中的同构化创新思想教育. 见：程序设计语言及其教学探索. 北京：清华大学出版社，2006. 169～172

6 喻敏，周启海. 论 VFP 程序设计教学中的同构化创新思想教育. 见：程序设计语言及其教学探索. 北京：清华大学出版社，2006. 173～176

（上接第 290 页）

例 2 图 6a)的网 N_5 也不是一个 M_ϕ-活网。如果规定对 N_5 只插入一个源变迁(每个源变迁只有一个后集库所)，那么不论怎样插入，所得到的新网也不是 M_ϕ-活网。若插入两个源变迁，那么共有 6 种插入方法。其中有 4 种插入所得到的新网是 M_ϕ-活网，另外 2 种插入得到的新网不是 M_ϕ-活网。图 6b)给出的只是其中一种合理的插入方法。

4 M_ϕ-可达网的特征

定义 2 设 $N=(S,T;F)$为一个网。如果对于网 N 的任意初始标识 M_0，在 $\Sigma=(N,M_0)$中都有 $M_\phi\in RM_0$)，则称 N 为一个 M_ϕ-可达网。

定理 4 网 $N=(S,T;F)$为一个 M_ϕ-可达网的充分必要条件是：1) $\exists t\in T: t^\cdot=\phi$；2) $\forall s\in S, \exists t_1\in T: t_1^\cdot=\phi$ 且存在从 s 到 t_1 的一条有向路 P 满足：$\forall t\in T:(s\in {}^\cdot t\vee|{}^\cdot t|\geqslant 2)\rightarrow t$ 不在有向路 P 上(证明方法类似于定理 1 和定理 2 的证明)。 □

如果一个网 N 不是 M_ϕ-可达网，可以对它施加插入操作，插入一些汇变迁(若 $t^\cdot=\phi$，则称 t 为一个汇变迁)，使之变成一个 M_ϕ-可达网。显而易见的做法是对 N 的每个库所 s，都加入一个汇变迁 t，使得 ${}^\cdot t=\{s\}$。诚然，这也不是必要条件。同样有这样的一个问题：怎样判定对任意给定的一个非 M_ϕ-可达网，至少要插入多少个汇编前(假设规定每个汇变迁都只有一个前集库所)，以及怎样插入才能使之变成一个 M_ϕ-可达网。这个问题尚待进一步研究。下面通过一个具体例子加以说明。

例 3 图 5a)的网 N_4 显然不是一个 M_ϕ-可达网。可以验证对于这个网，如果只插入一个汇变迁，那么不论怎样插入也不能变成 M_ϕ-可达网。如果最多只插入两个汇变迁，那么只有一种插入方式可以得到 M_ϕ-可达网，如图 7 所示。其它的插入方式都不可能得到 M_ϕ-可达网。

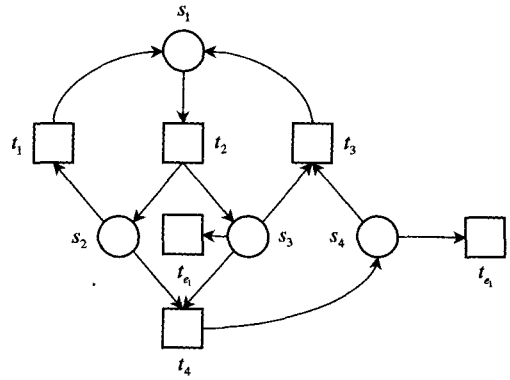

图 7 一个 M_ϕ-可达网 $N_4(+\{t_{e_1},t_{e_2}\})$

结束语 本文对 Petri 网的空标识进行了讨论，指出空标识不仅有正确的理论依据，而且也有深刻的实际背景。文中分别对空标识作为一个网系统的初始标识和空标识作为一个网的可达标识两种情况进行了分析，分别定义了 M_ϕ-活网和 M_ϕ-可达网，并得到了以下结果。

1)分别给出和证明了 M_ϕ-活网和 M_ϕ-可达网的结构特征。

2)证明了如果一个网系统的初始标识作为空标识 M_ϕ，那么这个网系统是活的当且仅当网中有变迁是一级活的。

3)当一个网不是 M_ϕ-活网时，可以在 N 中插入一些源变迁使之成为 M_ϕ-活的。

4)当一个网 N 不是 M_ϕ-可达网时，可以在 N 中插入一些汇变迁使之变成 M_ϕ-可达的。

本文也留下了两个公开问题：

1)怎样判定对一个非 M_ϕ-活网，至少要插入多少个源变迁，才能使之变成 M_ϕ-活网。

2)怎样判定对一个非 M_ϕ-可达网，至少要插入多少个汇变迁，才能使之变成 M_ϕ-可达网。

参 考 文 献

1 吴哲辉著. Petri 网导论. 北京：机械工业出版社，2006