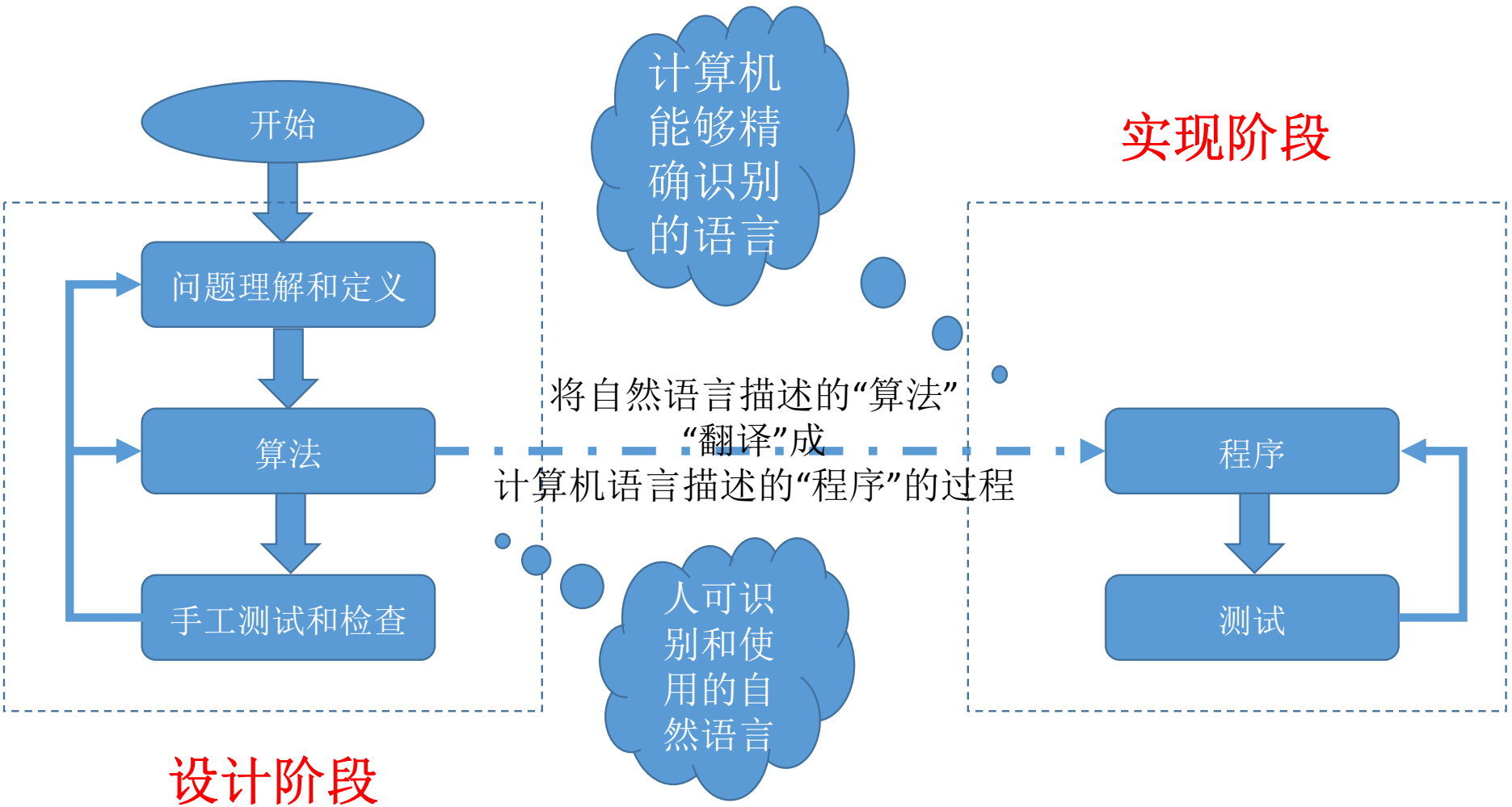
问题求解论题1-6
如何将算法告诉计算机

陶先平 陈道蓄

问题1

你如何将算法告诉计算机？

问题求解


开始
问题理解和定义
算法
手工测试和检查
设计阶段
计算机能够精确识别的语言
人可识别和使用的自然语言
将自然语言描述的“算法”“翻译”成计算机语言描述的“程序”的过程
实现阶段
程序
测试

请思考以下问题

- 在任何时候，编程之前必须先完成算法设计
  - Why？
- 计算机能够识别和处理的“语言”，不仅仅是C、C++等，也可以是自己定义的语言
  - For example？
- 程序运行出错，涉及到哪些内容？
  - 在“翻译”之前，尽量排除算法错误

问题2

计算机如何从二进制*bits*中识别并执行程序指令呢？

40年代的编程：

Before the middle of the 1940s,
computer operators
“hardwired” their programs

而后，二进制代码

显然，hardwired program不易修改，set switches不能算是编码！

如何让计算机的执行做到：programmable?
A big problem!

Big idea:

1,计算机提供基本的hardwired“原子操作”

2,提供编码方式，支持程序员组合“原子操作”，编写“程序”

3,将“程序”存放在存储空间中

4,计算机“解读”程序编码，执行原子操作

```
0010001000000100
0010010000000100
0001011001000010
0011011000000011
1111000000100101
0000000000000101
0000000000000110
0000000000000000
```

例如：

- 0010: 操作码,将”某个”内存中的数据复制到”某个”寄存器中
- 001: 001号寄存器
  - 第一个操作数
- 000000100：地址偏移量，4
  - 第二个操作数：从下一条指令所存储的地址向后偏移4个16位，取值5

执行效果：将5赋值给1号寄存器

```
0010001000000100
0010010000000100
0001011001000010
0011011000000011
1111000000100101
0000000000000101
0000000000000110
0000000000000000
```

问题3

你看到“编程”了吗？
你看到“语言”了吗？

符号语言

- The early programmers realized that it would be a tremendous help to ***use mnemonic symbols*** for the instruction codes and memory locations, so they developed **assembly language** for this purpose.

```
.ORIG x3000        ; Address (in hexadecimal) of the first instruction
LD   R1, FIRST     ; Copy the number in memory location FIRST to register R1
LD   R2, SECOND    ; Copy the number in memory location SECOND to register R2
ADD R3, R2, R1     ; Add the numbers in R1 and R2 and place the sum in
                   ; register R3
ST   R3, SUM       ; Copy the number in R3 to memory location SUM
HALT               ; Halt the program
FIRST   .FILL #5   ; Location FIRST contains decimal 5
SECOND .FILL #6    ; Location SECOND contains decimal 6
SUM     .BLKW #1   ; Location SUM (contains 0 by default)
.END               ; End of program
```

高级程序设计语言

- Programming language abstractions fall into two general categories: **data abstraction** and **control abstraction**.
  - Data abstractions simplify for human users the behavior and attributes of data, such as numbers, character strings, and search trees.
  - Control abstractions simplify properties of the transfer of control, that is, the modification of the execution path of a program based on the situation at hand. Examples of control abstractions are loops, conditional statements, and procedure calls.

```
int first = 5;
int second = 6;
int sum = first + second;
```

Data: Basic Abstractions

- Basic data abstractions in programming languages hide the internal representation of common data values in a computer
- Another basic data abstraction is the use of symbolic names to hide locations in computer memory that contain data values
  - int x;

Data: Structured Abstractions

- The **data structure** is the principal method for collecting related data values into a single unit.
  - Array: int a[10];
  - Record: ……

Data: Unit Abstractions

- In a large program, it is useful and even necessary to group related data and operations on these data together. Typically, such abstractions include access conventions and restrictions that support **information hiding**.
  - The unit abstraction is often associated with the concept of an **abstract data type**, broadly defined as a set of data values and the operations on those values.
  - Classes in C++ and Java

Control: Basic Abstractions

- Typical basic control abstractions are those statements in a language that combine a few machine instructions into an abstract statement that is easier to understand than the machine instructions.
- SUM = FIRST + SECOND

Control: Structured Abstractions

- Structured control abstractions divide a program into groups of instructions that are nested within tests that govern their execution. They, thus, help the programmer to express the logic of the primary control structures of sequencing, selection, and iteration (loops)

```
Iterator<String> iter = exampleList.iterator()
while (iter.hasNext())
    System.out.println(iter.next());
```

```
int sum = 0;
for (int i = 0; i < 10; i++){
    int data = list[i];
    if (data < 0)
        data = -data;
    sum += data;
}
```

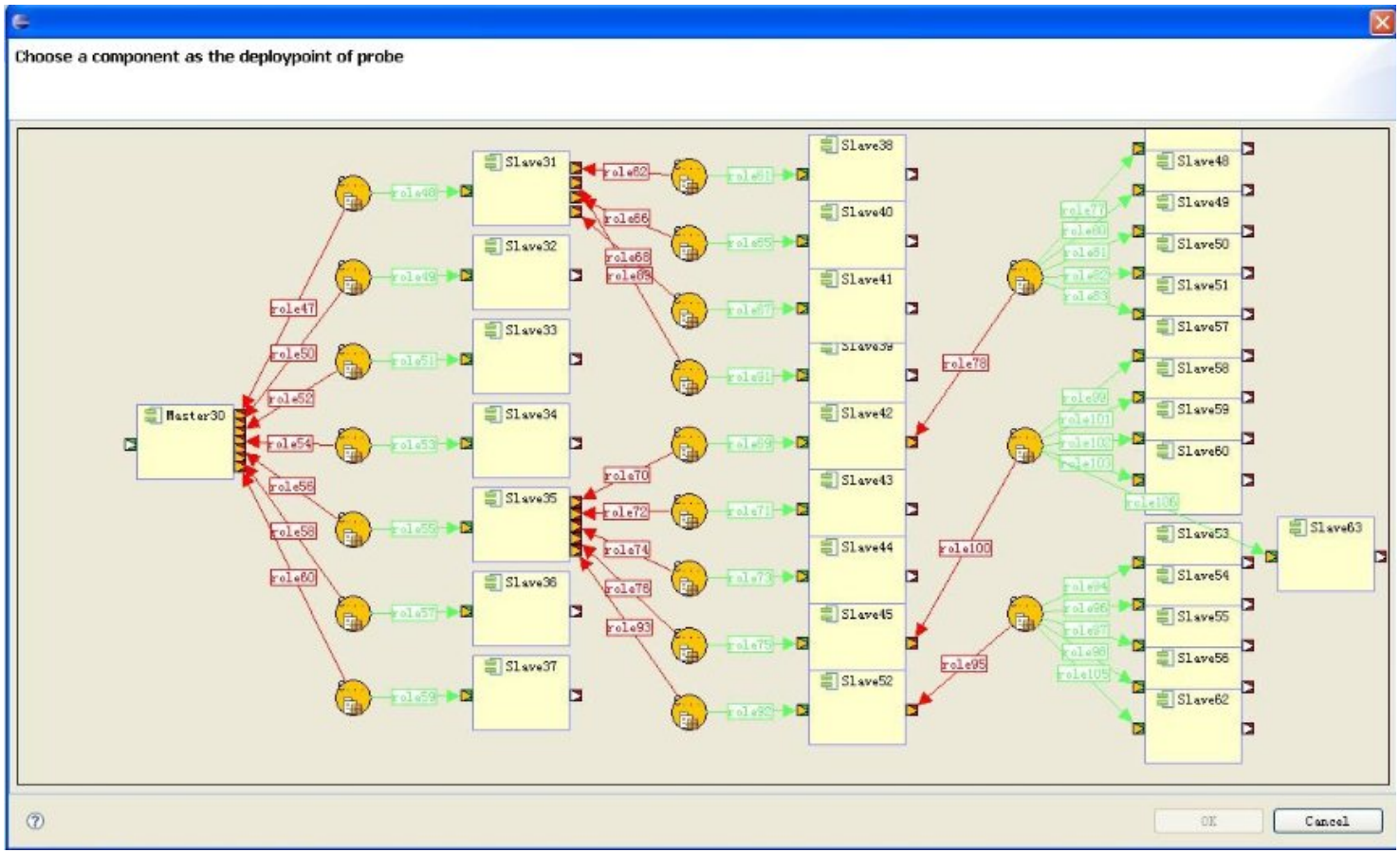
Control: Structured Abstractions

- Procedure：
  - This allows a programmer to consider a sequence of actions as a single action that can be called or invoked from many other points in a program

```
function gcd(u, v: in integer) return integer is
    begin
        if v = 0
            return u;
        else
            return gcd(v, u mod v);
        end if;
    end gcd;
```

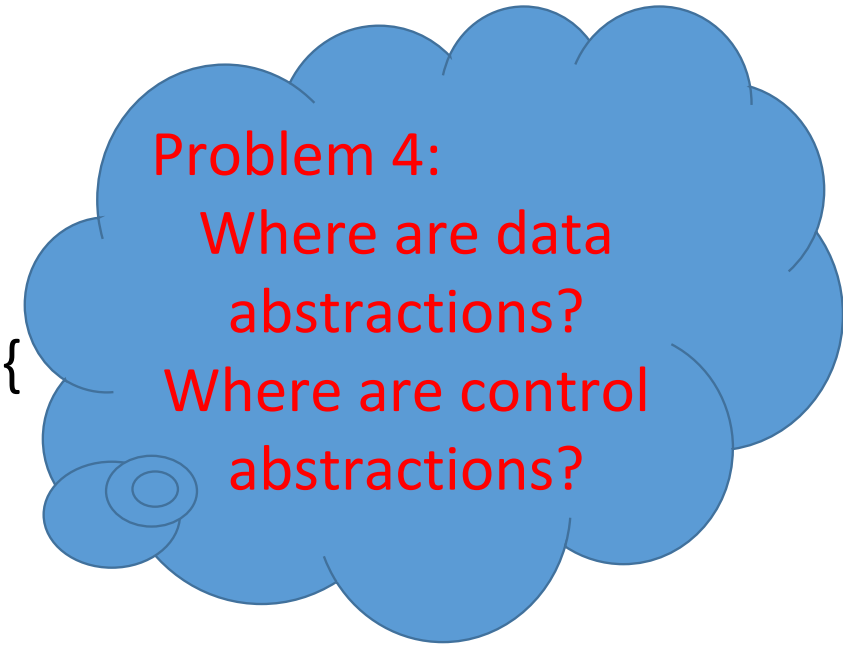
Control: Unit Abstractions

- Control can also be abstracted to include a collection of procedures that provide logically related services to other parts of a program and that form a **unit**, or stand-alone, part of the program



For example

```
int n=0;
grade = compute_grade[n];
while((grade<90)&&(n<number_of_students)){
    n++;
    grade=compute_grade[n];
}
if (grade>=90)
    cout <<“There is a student who got a score of ”<<grade <<endl;
else cout <<“No student has a high score”
```



Prolog：一种不同的设计风格

```
Predicates                                  /*谓词段，对要用的谓词名和参数进行说明*/
    likes(symbol, symbol)
    friend(symbol, symbol)
clauses                                     /*子句段，存放所有的事实和规则*/
    likes(bell,sports).                     /*前4行是事实*/
    likes(mary,music).
    likes(mary,sports).
    likes(jane,smith).
    friend(john,X):-likes(X,sports),likes(X,music).      /*本行是规则*/
 friend(john,X)
```

运行结果为：X=mary （mary是john的朋友）

问题5：

为什么会有不同的设计风格？

问题6：

如何做到编制出来的程序“精确”、“无歧义”？

语言的语法和语义

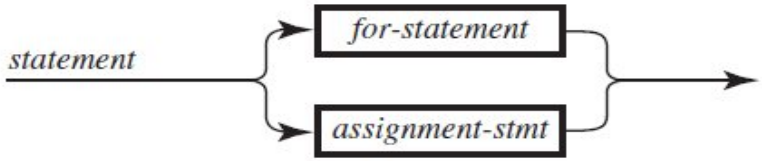
语法：规定了什么样的句子是合法的，进而规定了什么样的程序是合法的！

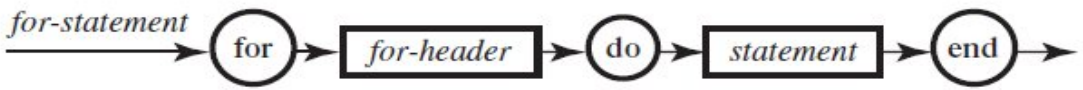
例如：

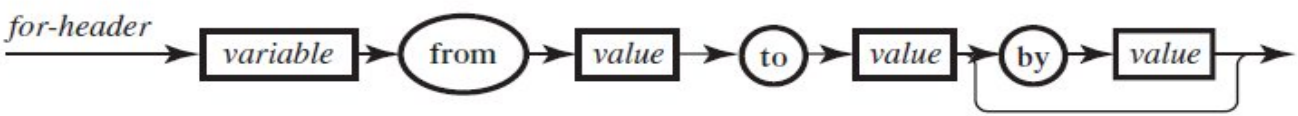
```
int sum=0;
int salary[100];
for index from 0 to 99 by 1 do
    sum = sum+salary[index];
end
```

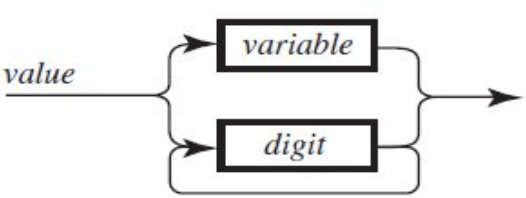
例如：

```
int sum=0;
int salary[100];
for index = 0 to 99 by 1 do
    sum = sum+salary[index];
end
```

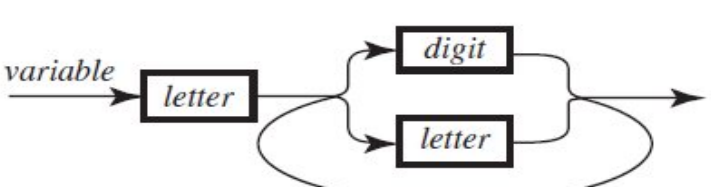








语法规定了程序的正确性吗？



语言的语法和语义

- 语义：用来精确定义一条语句乃至一个程序的准确意义

- 例：

```
#include "stdio.h"
main(){
   int x,y;
   x=3;
   y=x+(++x)+(++x);
   printf("%d,%d",x,y);
}
```

总结

- 算法设计和程序设计是问题求解的两个重要环节
  - 当然，数据结构的设计隐藏在两者中间
- 出于不同的目的，我们可以采纳不同“级别”和“风格”的程序设计语言进行编程
- 程序语言需要在“数据”和“控制”两个“线索”上提供“抽象机制”，供编程人员使用
  - 所谓抽象机制，可以具体理解为：语言设施
- 程序语言本身还涉及到语法和语义的描述