Specification and Abstraction

Learning Objectives

After learning the concepts in this chapter, the students will be able

- To understand the concept of algorithmic problem solving.
- To apply the knowledge of algorithmic technique in problem solving.

A number of processes performed in our daily life follow the step-by-step execution of a sequence of instructions. Getting ready to school in the morning, drawing "kolams", cooking a dish, adding two numbers are examples of processes. Pro-cesses are generated by executing algorithms. In this chapter, we will see how algorithms are specified and how elements of a process are abstracted in algorithms.

6.1 Algorithms

An algorithm is a sequence of instructions to accomplish a task or solve a problem. An instruction describes an action. When the instructions are executed, a process evolves, which accomplishes the intended task or solves the given problem. We can compare an algorithm to a recipe, and the resulting process to cooking.

We are interested in executing our algorithms in a computer. A computer can only execute instructions in a programming language. Instructions of a computer are also known as statements.Therefore, ultimately, algorithms must be expressed using statements of a programming language.

A problem is specified by given input data, desired output data and a relation between the input and the output data. An algorithm starts execution with the input data, executes the statements, and finishes execution with the output data. When it finishes execution, the specified relation between the input data and the output data should be satisfied. Only then has the algorithm solved the given problem.

DO YOU KNOW?

G Polya was a Hungarian mathematician. He made fundamental contributions to combinatorics, number theory, numerical analysis and probability theory. He is also noted for his work in heuristics and mathematics education, identifying systematic methods of problem solving to further discovery and invention in mathematics for students and teachers. In "How to Solve It", he suggests the following steps when solving a mathematical problem: 1. Understand the problem. 2. Devise a plan. 3. Carry out the plan. 4. Review your work.

An algorithm is a step by step sequence of statements intended to solve a problem. When executed with input data, it generates a computational process, and ends with output data, satisfying the specified relation between the input data and the output data.

Example 6.1. Add two numbers: To add two numbers, we proceed by adding the right most digits of the two numbers, then the next right most digits together with carry that resulted from the previous (right) position, and so on. The computational process for adding 2586 and 9237 is illustrated in Table 6.1.

Step	5	4	3	2	1
Carry	1	0	1	1	-
Number	1	2	5	8	6
Number	2	9	2	3	7
Sum	1	1	8	2	3

Table 6.1: The process for adding two numbers

6.2 Algorithmic Problems

There are some principles and techniques for constructing algorithms. We usually say that a problem is algorithmic in nature when its solution involves the construction of an algorithm. Some types of problems can be immediately recognized as algorithmic.

Example 6.2. Consider the well-known Goat, grass and wolf problem: A farmer wishes to take a goat, a grass bundle and a wolf across a river. However, his boat can take only one of them at a time. So several trips are necessary to across the river. Moreover, the goat should not be left alone with the grass (otherwise, the goat would eat the grass), and the wolf should not be left alone with the goat (otherwise, the wolf would eat the goat). How can the farmer achieve the task? Initially, we assume that all the four are at the same side of the river, and finally, all the four must be in the opposite side. The farmer must be in the boat when crossing the river. A solution consists of a sequence of instructions indicating who or what should cross. Therefore, this is an algorithmic problem. Instructions may be like

Let the farmer cross with the wolf.

or

Let the farmer cross alone.

However, some algorithmic problems do not require us to construct algorithms. Instead, an algorithm is provided and we are required to prove some of its properties.

Example 6.3. Consider The Chameleons of Chromeland problem: On the island of Chromeland there are three different types of chameleons: red chameleons, green chameleons, and blue chameleons. Whenever two chameleons of different colors meet, they both change color to the third color. For which number of red, green and blue chameleons it is possible to arrange a series of meetings that results in all the chameleons displaying the same color? This is an algorithmic problem, because there is an algorithm to arrange meetings between chameleons. Using some properties of the algorithm, we can find out for which initial number of chameleons, the goal is possible.

6.3 Building Blocks of Algorithms

We construct algorithms using basic building blocks such as

- Data
- Variables
- Control flow
- Functions

6.3.1 Data

Algorithms take input data, process the data, and produce output data. Computers provide instructions to perform operations on data. For example, there are

instructions for doing arithmetic operations on numbers, such as add, subtract, multiply and divide. There are different kinds of data such as numbers and text.

6.3.2 Variables

Variables are named boxes for storing data. When we do operations on data, we need to store the results in variables. The data stored in a variable is also known as the value of the variable. We can store a value in a variable or change the value of variable, using an assignment statement.

Computational processes in the real-world have state. As a process evolves, the state changes. How do we represent the state of a process and the change of state, in an algorithm? The state of a process can be represented by a set of variables in an algorithm. The state at any point of execution is simply the values of the variables at that point. As the values of the variables are changed, the state changes.

Example 6.4. State: A traffic signal may be in one of the three states: green, amber, or red. The state is changed to allow a smooth flow of traffic. The state may be represented by a single variable signal which can have one of the three values: green, amber, or red.

6.3.3 Control flow

An algorithm is a sequence of statements. However, after executing a statement, the next statement to be executed need not be the next statement in the algorithm. The statement to be executed next may depend on the state of the process. Thus, the order in which the statements are executed may differ from the order in which they are written in the algorithm. This order of execution of statements is known as the control flow.

There are three important control flow statements to alter the control flow depending on the state.

- In sequential control flow, a sequence of statements are executed one after another in the same order as they are written.
- In alternative control flow, a condition of the state is tested, and if the condition is true, one statement is executed; if the condition is false, an alternative statement is executed.
- In iterative control flow, a condition of the state is tested, and if the condition is true, a statement is executed. The two steps of testing the condition and executing the statement are repeated until the condition becomes false.

6.3.4 Functions

Algorithms can become very complex. The variables of an algorithm and dependencies among the variables may be too many. Then, it is difficult to build algorithms correctly. In such situations, we break an algorithm into parts, construct each part separately, and then integrate the parts to the complete algorithm.

The parts of an algorithm are known as functions. A function is like a sub algorithm. It takes an input, and produces an output, satisfying a desired input output relation.

Example 6.5. Suppose we want to calculate the surface area of a cylinder of radius r and height h.

$$A = 2\pi r^2 + 2\pi rh$$

We can identify two functions, one for calculating the area of a circle and the other for the circumference of the circle. If we abstract the two functions as circle_area(r) and circle_circumference(r), then cylinder_area(r, h) can be solved as

cylinder_area (r,h) = 2 X circle_area (r) + circle_circumference (r) X h

6.4 Algorithm Design Techniques

There are a few basic principles and techniques for designing algorithms.

1. **Specification:** The first step in problem solving is to state the problem precisely. A problem is specified in terms of the input given and the output desired. The specification must also state the properties of the given input, and the relation between the input and the output.
2. **Abstraction:** A problem can involve a lot of details. Several of these details are unnecessary for solving the problem. Only a few details are essential. Ignoring or hiding unnecessary details and modeling an entity only by its essential properties is known as abstraction. For example, when we represent the state of a process, we select only the variables essential to the problem and ignore inessential details.
3. **Composition:** An algorithm is composed of assignment and control flow statements. A control flow statement tests a condition of the state and, depending on the value of the condition, decides the next statement to be executed.
4. **Decomposition:** We divide the main algorithm into functions. We construct each function independently of the main algorithm and other functions. Finally, we construct the main algorithm using the functions. When we use the functions, it is enough to know the specification of the function. It is not necessary to know how the function is implemented.

6.5 Specification

To solve a problem, first we must state the problem clearly and precisely. A problem is specified by the given input and the desired output. To design an algorithm for solving a problem, we should know the properties of the given input and the properties of the desired output. The goal of the algorithm is to establish the relation between the input and the desired output.

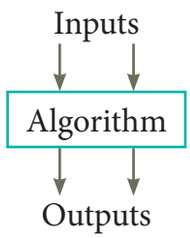


Figure 6.1: Input-output relation

An algorithm is specified by the properties of the given input and the relation between the input and the desired output. In simple words, specification of an algorithm is the desired input-output relation.

The inputs and outputs are passed between an algorithm and the user through variables. The values of the variables when the algorithm starts is known as the initial state, and the values of the variables when the algorithm finishes is known as the final state.

Let P be the required property of the inputs and Q the property of the desired outputs. Then the algorithm S is specified as

1. algorithm_name (inputs)
2. -- inputs : P
3. -- outputs: Q

This specification means that if the algorithm starts with inputs satisfying P, then it will finish with the outputs satisfying Q.

A double dash -- indicates that the rest of the line is a comment. Comments are statements which are used to annotate a program for the human readers and not executed by the computer. Comments at crucial points of flow are useful, and even necessary, to understand the algorithm. In our algorithmic notation, we use double dashes (—) to start a comment line. (In C++, a double slash // indicates that the rest of the line is a comment).

Example 6.6. Write the specification of an algorithm to compute the quotient and remainder after dividing an integer A by another integer B. For example,

divide (22, 5) = 4, 2

divide (15, 3) = 5 , 0

Let A and B be the input variables. We will store the quotient in a variable q and the remainder in a variable r. So q and r are the output variables.

What are the properties of the inputs A and B?

1. A should be an integer. Remainder is meaningful only for integer division, and
2. B should not be 0, since division by 0 is not allowed.

We will specify the properties of the inputs as

— inputs: A is an integer and B ≠ 0

What is the desired relation between the inputs A and B, and the outputs q and r?

1. The two outputs q (quotient) and r (remainder) should satisfy the property

   A = q X B + r, and
2. The remainder r should be less than the divisor B,

   0 ≤ r < B

Combining these requirements, we will specify the desired input-output relation as

— outputs: A = q X B + r and 0 < r < B.

The comment that starts with — inputs: actually is the property of the given inputs. The comment that starts with — outputs: is the desired relation between the inputs and the outputs. The specification of the algorithm is

1. **divide (A , B)**
2. **-- inputs: A is an integer and B ≠ 0**
3. **-- outputs : A = q X B + r and 0 ≤ r < B**

Specification format: We can write the specification in a standard three part format:

- The name of the algorithm and the inputs.
- Input: the property of the inputs.
- Output: the desired input-output relation.

The first part is the name of the algorithm and the inputs. The second part is the property of the inputs. It is written as a comment which starts with — inputs: The third part is the desired input-output relation. It is written as a comment which starts with — outputs:. The input and output can be written using English and mathematical notation.

Example 6.7. Write the specification of an algorithm for computing the square root of a number.

1. Let us name the algorithm square_root.
2. It takes the number as the input. Let us name the input n. n should not be negative.
3. It produces the square root of n as the output. Let us name the output y. Then n should be the square of y.

Now the specification of the algorithm is

square_root(n)

-- inputs: n is a real number, n ≥ 0.

-- outputs: y is a real number such that $y^2 = n$.

6.5.1 Specification as contract

Specification of an algorithm serves as a contract between the designer of the algorithm and the users of the algorithm, because it defines the rights and responsibilities of the designer and the user.

Ensuring that the inputs satisfy the required properties is the responsibility of the user, but the right of the designer. The desired input-output relation is the responsibility of the designer and the right of the user. Importantly, if the user fails to satisfy the properties of the inputs, the designer is free from his obligation to satisfy the desired input-output relation.

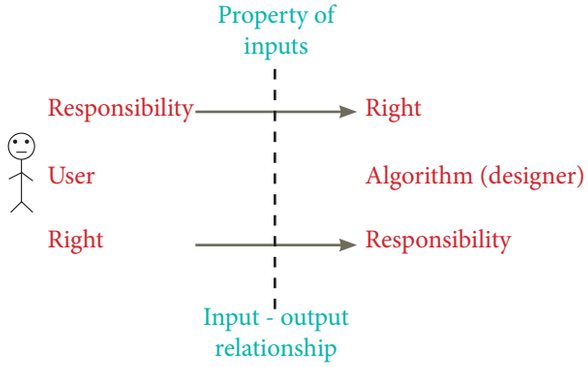


Figure 6.2: Input property and the input-output relation as rights and responsibilities

Example 6.8. Consider the specification of the algorithm square_root.

square_root(n)

-- inputs: n is a real number, n ≥ 0.

-- outputs : y is a real number such that $y^2 = n$.

The algorithm designer can assume that the given number is non-negative, and construct the algorithm. The user can expect the output to be the square root of the given number.

The output could be the negative square root of the given number. The specification did not commit that the output is the positive square root. If the user passes a negative number as the input, then the output need not be the square root of the number.

6.6 Abstraction

To ride a bicycle, it is sufficient to understand the functioning of the pedal, handlebar, brakes and bell. As a rider, we model a bicycle by these four features. A bicycle has a lot more details, which the rider can ignore. Those details are irrelevant for the purpose of riding a bicycle.

A problem can involve a lot of details. Several of these details are irrelevant for solving the problem. Only a few details are essential. Abstraction is the process of ignoring or hiding irrelevant details and modeling a problem only by its essential features. In our everyday life, we use abstractions unconsciously to handle complexity. Abstraction is the most effective mental tool used for managing complexity. If we do not abstract a problem adequately, we may deal with unnecessary details and complicate the solution.

Example 6.9. A map is an abstraction of the things we find on the ground. We do not represent every detail on the ground. The map-maker picks out the details that we need to know. Different maps are drawn for different purposes and so use different abstractions, i.e., they hide or represent different features. A road map is designed for drivers. They do not usually worry about hills so most hills are ignored on a road map. A walker's map is not interested in whether a road is a one-way street, so such details are ignored.

Example 6.10. In medicine, different specialists work with different abstractions of human body. An orthopaedician works with the abstraction of skeletal system, while a gastroenterologist works with digestive system. A physiotherapist abstracts the human body by its muscular system.

We use abstraction in a variety of ways while constructing algorithms — in the specification of problems, representing state by variables, and decomposing an algorithm to functions. An algorithm designer has to be trained to recognize which features are essential to solve the problem, and which details are unnecessary. If we include unnecessary details, it makes the problem and its solution over-complicated.

Specification abstracts a problem by the properties of the inputs and the desired input-output relation. We recognize the properties essential for solving the problem, and ignore the unnecessary details.

State: In algorithms, the state of a computation is abstracted by a set of variables.

Functions: When an algorithm is very complex, we can decompose it into functions and abstract each function by its specification.

6.6.1 State

State is a basic and important abstraction. Computational processes have state. A computational process starts with an initial state. As actions are performed, its state changes. It ends with a final state. State of a process is abstracted by a set of variables in the algorithm. The state at any point of execution is simply the values of the variables at that point.

Example 6.11. Chocolate Bars: A rectangular chocolate bar is divided into squares by horizontal and vertical grooves. We wish to break the bar into individual squares.

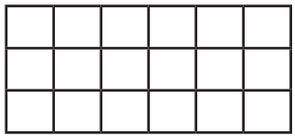
To start with, we have the whole of the bar as a single piece. A cut is made by choosing a piece and breaking it along one of its grooves. Thus a cut divides a piece into two pieces. How many cuts are needed to break the bar into its individual squares?

In this example, we will abstract the essential variables of the problem. We solve the problem in Example 8.6.

Essential variables: The number of pieces and the number of cuts are the essential variables of the problem. We will represent them by two variables, p and c, respectively. Thus, the state of the process is abstracted by two variables p and c.

Irrelevant details:

1. The problem could be cutting a chocolate bar into individual pieces or cutting a sheet of postage stamps into individual stamps. It is irrelevant. The problem is simply cutting a grid of squares into individual squares.



2. The sequence of cuts that have been made and the shapes and sizes of the resulting pieces are irrelevant too. From p and c, we cannot reconstruct the sizes of the individual pieces. But, that is irrelevant to solving the problem.

Example 6.12. Consider Example 6.2, Goat, grass and wolf problem. In this example,

we will write a specification of the problem. We will solve it in Example 7.1. The problem involves four individuals, and each is at one of the two sides of the river. This means that we can represent the state by four variables, and each of them has one of the two values. Let us name the variables as farmer, goat, grass and wolf, and their possible values L and R. A value of L means "at the left side". A value of R means "at the right side". Since the boat is always with the farmer, it is not important to introduce a variable to represent its position.

In the initial state, all four variables farmer, goat, grass, wolf have the value L.

farmer, goat, grass, wolf = L, L, L, L

In the final state, all four variables should have the value R.

farmer, goat, grass, wolf = R, R, R, R

The specification of the problem is

cross_river

-- **inputs: farmer, goat, grass, wolf = L, L, L, L**

-- **outputs: farmer, goat, grass, wolf = R, R, R, R**

subject to the two constraints that

1. the goat cannot be left alone with the grass:

   if goat = grass then farmer = goat

2. the goat cannot be left alone with the wolf:

   if goat = wolf then farmer = goat

6.6.2 Assignment statement

Variables are named boxes to store values. Assignment statement is used to store a value in a variable. It is written with the variable on the left side of the assignment operator and a value on the right side.

variable := value

When this assignment is executed, the value on the right side is stored in the variable on the left side. The assignment

m := 2

stores value 2 in variable m.

m

2

If the variable already has a value stored in it, assignment changes its value to the value on the right side. The old value of the variable is lost.

The right side of an assignment can be an expression.

variable := expression

In this case, the expression is evaluated and the value of the expression is stored in the variable. If the variable exists in the expression, the current value of the variable is used in evaluating the expression, and then the variable is updated. For example, the assignment

m := m + 3

evaluates the expression m + 3 using the current value of m.

m + 3
= 2 + 3
= 5

and stores the value 5 in the variable m.

m
5

The two sides of an assignment statement are separated by the symbol :=, known as assignment operator, and read as "becomes" or "is assigned". The assignment statement

v := e

is read as v "becomes" e. Note that assignment operator is not equality operator[1]. The meanings of v := e and v = e are different. Assignment does not state a mathematical equality of a variable, but changes the value of a variable. The assignment m := m + 3 does not state that m is equal to m + 3. Rather, it changes the value of the variable m to the value of the expression m + 3.

An assignment statement can change the values of multiple variables simultaneously. In that case, the number of variables on the left side should match the number of expressions on the right side. For example, if we wish to assign to three variables v1, v2 and v3, we need 3 expressions, say, el, e2, e3.

vl, v2, v3 := el, e2 , e3

The left side is a comma-separated list of variables. The right side is a comma-separated list of expressions. To execute an assignment statement, first evaluate all the expressions on the right side using the current values of the variables, and then store them in the corresponding variables on the left side.

Example 6.13. What are the values of variables m and n after the assignments in line (1) and line (3)?

1. **m, n := 2 , 5**
2. **-- m, n = ? , ?**
3. **m,n:=m+3,n-1**
4. **-- m, n = ? , ?**

The assignment in line (1) stores 2 in variable m, and 5 in variable n.

m	n
2	5

The assignment in line (3) evaluates the expressions m + 3 and n - 1 using the current values of m and n as

m + 3 , n - 1

=2+3,5-1

= 5,4

and stores the values 5 and 4 in the variables m and n, respectively.

m	n
5	4

1. **m, n := 2,5**
2. **-- m, n = 2 , 5**
3. **m, n := m + 3, n - 1**
4. **-- m, n = 2 + 3, 5-1 = 5, 4**

Values of the variables after the two assignments are shown in in line (2) and line(4).

Example 6.14. In Example 6.11, we abstracted the state of the process by two

---

^Unfortunately, several programming languages, including C++, use the symbol = as assignment operator, and therefore, another symbol, == as equality operator.

variables p and c. The next step is to model the process of cutting the chocolate bar. When we make a single cut of a piece, the number of pieces (p) and the number of cuts (c) both increase by 1. We can model it by an assignment statement.

p, c := p + 1, c+1

Points to Remember:

which is read as p and c "become" p + 1 and c + 1, respectively.

- A programming language provides basic statements and a notation for composing compound statements.
- An algorithm is a step-by-step sequence of statements to solve a problem.
- As an algorithm is executed, a process evolves which solves the problem.
- Algorithmic problem solving involves construction of algorithms as well as proving properties of algorithms.
- The specification of an algorithm consists of the name of the algorithm (together with its inputs), the input property, and the desired input-output relation.
- Specification of an algorithm is a contract between the designer and users of the algorithm.
- Abstraction is the process of hiding or ignoring the details irrelevant to the task so as to model a problem only by its essential features.
- Specification abstracts a problem by the essential variables of the problem.
- The values of the variables in an algorithm define the state of the process.
- Assignment statement changes the values of variables, and hence the state.

Evaluation

SECTION – A

Choose the correct answer

1. Which of the following activities is algorithmic in nature?

   (a) Assemble a bicycle. (b) Describe a bicycle.

   (c) Label the parts of a bicycle. (d) Explain how a bicycle works.

2. Which of the following activities is not algorithmic in nature?

   (a) Multiply two numbers. (b) Draw a kolam.

   (c) Walk in the park. (d) Swaping of two numbers.

3. Omitting details inessential to the task and representing only the essential features of the task is known as

   (a) specification (b) abstraction (c) composition (d) decomposition

4. Stating the input property and the input-output relation a problem is known

   (a) specification (b) statement (c) algorithm (d) definition

5. Ensuring the input-output relation is

   (a) the responsibility of the algorithm and the right of the user.

   (b) the responsibility of the user and the right of the algorithm.

   (c) the responsibility of the algorithm but not the right of the user.

   (d) the responsibility of both the user and the algorithm.

6. If i = 5 before the assignment i := i-1 after the assignment, the value of i is

   (a) 5 (b) 4 (c) 3 (d) 2

7. If 0 < i before the assignment i := i-1 after the assignment, we can conclude that

   (a) 0 < i (b) 0 ≤ i (c) i = 0 (d) 0 ≥i

SECTION-B

Very Short Answers

1. Define an algorithm.
2. Distinguish between an algorithm and a process.
3. Initially,

   farmer, goat, grass, wolf = L, L, L, L

   and the farmer crosses the river with goat. Model the action with an assignment statement.
4. Specify a function to find the minimum of two numbers.
5. If $\sqrt{2}$ = 1.414, and the square_root() function returns -1.414, does it violate the following specification?

   -- square_root (x)

   -- inputs: x is a real number , x ≥ 0

   -- outputs: y is a real number such that $y^2=x$

SECTION-C

Short Answers

1. When do you say that a problem is algorithmic in nature?
2. What is the format of the specification of an algorithm?
3. What is abstraction?
4. How is state represented in algorithms?
5. What is the form and meaning of assignment statement?
6. What is the difference between assignment operator and equality operator?

SECTION - D

Explain in detail

1. Write the specification of an algorithm hypotenuse whose inputs are the lengths of the two shorter sides of a right angled triangle, and the output is the length of the third side.

2. Suppose you want to solve the quadratic equation ax2 + bx + c = 0 by an algorithm.

   quadratic_solve (a, b, c)

   -- inputs : ?

   -- outputs: ?

   You intend to use the formula and you are prepared to handle only real number roots. Write a suitable specification.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

3. Exchange the contents: Given two glasses marked A and B. Glass A is full of apple drink and glass B is full of grape drink. For exchanging the contents of glasses A and B, represent the state by suitable variables, and write the specification of the algorithm.

Books

1. Roland Backhouse, Algorithmic Problem Solving, John Wiley & Sons Ltd, 2011.

2. Krysia Broda, Susan Eisenbach, Hessam Khoshnevisan, Steve Vickers, Reasoned Programming, Prentice Hall, 1994