

## Representation and Implementation

An abstract data type (ADT) is a set of values together with a collection of operations on those values. The operations are as important as the values. The word 'abstract' means the values are viewed independently of their representation. In computing this means the values are regarded as independent of a machine or language. The integers together with the usual arithmetic operations form a mathematical ADT.

The Egyptians (~2000BC) had a practical representation for natural numbers together with algorithms for multiplication and division by doubling and halving. (Examples in lecture.)

The Old Babylonians (~1800BC) had a floating-point, place value notation with sophisticated procedural 'algorithms' for problem solving. (As in lectures.)

The abacus (or counting board) gives another way of representing natural numbers and so does the Hindu-Arabic decimal notation that we use today.

For each of these *representations* of numbers there is a corresponding *implementation* of the arithmetic operations. Often there is a trade-off between the work involved in the representation, and in the implementation, of an ADT.

## **Simplifying Solutions by transforming the Problem**

A place-value notation simplified arithmetic.

Transforming problems from one representation to another is a powerful problem-solving technique.

Another simple example in mathematics is that of the logarithm function which transforms a multiplication into an addition:

$$\exp(\log a + \log b) = a \times b$$

The transformation of a matrix of coefficients into an equivalent triangular matrix helps to solve a set of simultaneous equations.

A list that is sorted allows for a more efficient binary search algorithm. Another example arises from the analysis of the following 2-person game.

### **The 31-game**

A die is thrown normally to start with, then two players take turns to turn the die about an edge on to one of its 4 vertical faces to produce a new value on the top face. A running total is kept for each player of the values displayed on the top face. The first player forced to make the total exceed 31 loses.

If the players could make a complete analysis of the possibilities, how would they play? (They wouldn't play at all! Why?)

## Evaluation of the 31-game

Consider the game tree for the 31-game. For every position there are four possible moves. Any sequence of moves is represented by a path through the tree. The tree is a geometric representation of the set of positions (nodes) subject to the relation “can directly lead to” (edges).

Imagine the whole tree has been drawn. Positions where total exceeds 30 are the leaves. Define an evaluation function from the set of nodes to  $\{0,1\}$  so that 0 represents ‘lost’ and 1 ‘won’. That is

valuation (node) = 0 iff player to move loses

Leaves represent position where total  $\geq 31$ . A node with total = 31 has value 0, a node with total  $>31$  has value 1 (the ‘player to move’ has just won).

Cut off leaves of tree and consider a ‘new’ leaf. All successors of the new leaf in old tree were valued 0 or 1. The value of the new leaf is 1 if and only if one of its successors had value 0. Continue up to the root of the tree to give a complete evaluation of the game tree.

In fact many nodes are equivalent in this tree. What matters is only the current total and the last move. So a better evaluation is of position pairs (total, lastMove) in an array.

Write a Java program that will evaluate such an array and use it to play the game optimally.

Extracts from the 31-game evaluation tree of (total, lastMove) pairs for game positions

