Abstract Data Types

A useful tool for specifying logical properties of a datatype is ADT.

To illustrate the concept of an ADT, consider the ADT RATIONAL, which corresponds to the mathematical concept of a rational number.

A rational number is a number that can be expressed as the quotient of two integers.

The operations on rational numbers are:
→ creation of a rational number from two integers
→ addition of two rational numbers
→ multiplication of two rational numbers
→ testing for equality between two rational numbers

Initial Specification of ADT

```
/* Value definition */
abstract typedef <integer, integer> RATIONAL;
condition RATIONAL[ ] != 0;

/* Operator definition */

abstract RATIONAL makerational(a, b)
int a, b;
precondition b != 0;
postcondition makerational[0] = a;
postcondition makerational[1] = b;

abstract RATIONAL add(a, b)   /* a + b */
RATIONAL a, b;
postcondition add[1] = a[1] * b[1];
postcondition add[0] = a[0] * b[1] + b[0] * a[1];

abstract RATIONAL mult(a, b)   /* a * b */
RATIONAL a, b;
postcondition mult[0] = a[0] * b[0];
postcondition mult[1] = a[1] * b[1];

abstract equal(a, b)   /* a == b */
RATIONAL a, b;
postcondition equal = (a[0] * b[1] == b[0] * a[1]);
```

An ADT consists of two parts: a value definition and an operator definition.

The value definition defines the collection of values for the ADT and consists of two parts: a definition clause and a condition clause.
For ex: the value definition for the ADT RATIONAL states that a RATIONAL value consists of two integers, the second of which does not equal 0.
The keyword abstract typedef introduces a value definition and the keyword condition is used to specify any conditions on the newly defined type.

An operator definition, each operator is defined as an abstract function with three parts: a header, the optional preconditions and the postconditions.
The postcondition specifies what the operation does. In a postcondition, the name of the function (in this example, mult) is used to denote the result of the function. Thus mult[0] and mult[1] represents numerator and denominator respectively.

The specification for addition (add) is:

$$\frac{a0}{a1} + \frac{b0}{b1} = \frac{a0 \times b1 + a1 \times b0}{a1 \times b1}$$

The creation operation (make rational) creates a rational number from two integers.

Preconditions specify any restrictions that must be satisfied before the operation can be applied.

In general, any two values in an ADT are equal if and only if the values of their components are equal.
For some data types, two values with unequal components may be considered equal.

For eg: the rational numbers 1/2, 2/4, 3/6, and 18/36 are equal despite the inequality of their components.

Another way of testing for rational equality is to check whether the cross products are equal.

Array as an ADT

```
abstract typedef <<eltype, ub>> ARRTYPE (ub, eltype);
condition type(ub) == int;
abstract eltype extract(a, i)
ARRTYPE (ub, eltype) a;
int i;
precondition 0 <= i < ub;
```

```
postcondition extract == a;

abstract store(a, i, elt)
ARRTYPE (ub, eltype) a;
int i;
eltype elt;
precondition 0 <= i < ub;
postcondition a[i] == elt;
```

Stack as an ADT

```
abstract typedef <<eltype>> STACK (eltype);
abstract empty(S)
STACK (eltype) S;
postcondition empty == (len(S) == 0);

abstract eltype pop(S)
STACK (eltype) S;
precondition empty(S) == FALSE;
postcondition pop == first(S');
S == sub(S', 1, len(S') - 1);

abstract push(S, elt)
STACK (eltype) S;
eltype elt;
postcondition S == <elt> + S'
```