ALGEBRAIC SPECIFICATION LANGUAGE

The algebraic specification language (ASL) is intended to provide the formal structure needed for modular programming and the flexibility needed for unencumbered design.

The main advantage of ASL is that it makes it simple to understand both what the machine is doing and what the designer is doing. Computation can be thought of as algebraic match-and-substitute. Specification can be thought of as construction of mathematical systems.

This language does not provide a visual metaphor or any intuitive user interface tools. These design tools can be built on top of the ASL, forming another layer which uses the functionality in the algebraic specification.

In Computer Science, this approach is called abstract data types. Specifying worlds can also be thought of as creating objects in object-oriented programming. In mathematics, using ASL is building a formal algebraic theory. In logic, it is Predicate Calculus with equality.

Reference: Bergstra, Heering and Klint (eds), Algebraic Specification, ACM Press: 1989.

Caution: This approach is under development, is likely to change, and can benefit from your critical analysis.

THE LANGUAGE OF ALGEBRAIC WORLDS

A ***world*** consists of a set of equations:

```
      World  ::=  { equation ... }
```

An ***equation*** is an identity relation between two terms:

```
      Equation  ::=  ( Term = Term )
```

A ***term*** is an expression which can be constructed from the labels and connectives in a language. A term is either a constant, a variable, a function-label, a relation-label, a set, or an equation:

```
      Term ::= constant | variable | function | relation | set | equation
```

A ***constant*** is a label, labels are typographic strings:

```
      Constant  ::=  label
```

A ***variable*** is a label with one or more underscores:

```
Variable  ::=  label_
```

A ***function-label*** is a labeled collection of variables:

```
Function  ::=  functionlabel[variables__]
```

A ***relation-label*** is also a labeled collection of variables:

```
Relation ::=  relationlabel[ variables__ ]
```

A ***set*** is a collection of terms bounded by curly brackets.

```
Set  ::=  {term ...}
```

Using this language assures that the designed world can be programmed. The template of what belongs in a world description follows.

THE STRUCTURE OF AN ALGEBRAIC SPECIFICATION

The name of the particular world is declared to be equivalent to the set of equations which define that world.

```
WorldName = { world-equations ... }
```

Include the following types of equations in every World:

sets	(collections of elementary elements or things)
functions	(transformations of things)
relations	(Boolean assertions about things)
rules	(equations which define the behavior of a world)

* Set names identify a collection of ground constants, the elementary objects in the world. Each label denotes a thing in the designed world.

```
( SETNAME = { label1 ... } )
```

* Functions transform things and the characteristics of things. How a function changes a constant is specified by a base case equation.

```
      ( functionlabel[ constant ] = term-which-reduces-to-a-constant  )
```

The general case of an function specifies how that function changes collections of things.

```
      ( functionlabel[ variablepatterns__ ] = term )
```

Functions can be recursive. Constants are functions with no arguments.

* Relations establish associations between collections of things.

```
      ( relationlabel[ variablepatterns__ ] = boolean-term )
```

* Equations express an equivalence between terms (structures in the world). Equations define the axioms, properties, or rules for the world; they specify how the world works.

```
      ( term-with-variables = term-without-variables  )
```

SOME TOOLS

The ASL is easier to use when some tools are included.

Mathematical Operators:

Logic, arithmetic, and set theory can be assumed to be available. The logical connectives {not or and if} permit complex Boolean relations and permit control structure to be embedded in equations. Arithmetic {+ - * % ^ < >} provides counting and numerical calculation. Sets can be composed and decomposed using set operators { union intersection complement }.

Generators:

The ellipsis ... can be thought of as a unique label generator. Rather than naming all the blocks for example, BLOCK = { block1 block2 block3 }, you can name the pattern for block names, BLOCK = { block1 ...}. The ellipsis can be constrained; BLOCK = { block1 ... block10 } will generate ten block names.

Indices:

It is often useful to index items in a set. The dot notation indicates parts of a complex term:

element1.part3	identifies part3 of element1.

An index notation can be used to identify a specific arbitrary element:

element:i	identifies an arbitrary single element i selected from the set of elements, without replacement.

Indices can also identify one element selected from a set:

element:1	selects one element, with replacement.

Set functions:

The ASL permits functions to operate on sets. The pattern variable with a double underbar, label__, specifies one or more members of a set which match the pattern. A set calculus allows sets to be treated as variables: CAPITALX = {a ...} labels the set itself, smallx = {a ...} labels the collection of elements.

SET1 = SET2	declares set equality
set1 = {a b c}	associates the label with an arbitrary element

Operations on capital-letter sets can be assumed to apply over the entire set.

SET1 + 5	adds 5 to each element of set1
+[SET1]	adds the elements of SET1

Typing:

The SETLABEL can be thought of as the type of the elements in a set. The type of constants and variables can be indicated after the label:

x_BOOL	x belongs to the set {true false}
b.BLOCK	b labels a block

Natural notation:

The pattern language lets us determine the form of expressions:

```
( block1 Is On block2 ) = On[block1 block2]
( if (p = true) then (q = true) ) = (If[p, q ] = true)
```

Sequences:

The elements of a set are separated by spaces, and in the case of ambiguity, by parentheses. The elements of a sequence are separated by commas.

{a b c}	a set
{a, b, c}	a sequence
and[a b c]	a set of arguments
if[a, b]	a sequence of arguments

Delimiters:

parentheses	(a=b)	for clarity
brackets	fn[a b]	for arguments
curly braces	{a b}	for sets

Using other worlds:

The USE command takes world labels as arguments and automatically includes the equations which define that world. USE is declared equal to a set of labels which identify the used objects.

USE[UnitVector] = {i j}	uses two unit vectors