Abstract Data Structure: SETS

Sets (unordered collections of unique objects) are a fundamental mathematical concept. Pure set operations are impossible to implement on a serial processor. Note that the model of sets is isomorphic with the model of *propositional calculus*, with the membership operator added.

A **set implementation** with the functions Insert, Delete, and Member is called a *dictionary*.

Mathematical model:

```
S = {x| <statement about x>}   extensional, defined by common property
S = {a,b,c,...}                intensional, defined by naming the members
```

empty set:	`not (x in S)  forall x`
membership:	`x in S  =def=  x=s1 or x=s2 or x=s3 or ...`
subset:	`if (x in S1) then (x in S2)`
union:	`(x in S1) or (x in S2)`
intersection:	`(x in S1) and (x in S2)`
difference:	`(x in S1) and not(x in S2)`

recursive set membership:

```
x in S  =def=
      not[x=empty-set] and (x = get-one[S] or (x in rest[S]))
```

Implementation functions:

```
Make-empty-set
Make-set[elements]
Insert[element,set]
Delete[element,set]
Equal[set1,set2]
```

`Cardinality[set]` = count of members

Characteristic function F:

```
(F[x] = 1 iff x in S)  and  (F[x] = 0 iff not(x in S))
```

Implementation using Enumeration: the named members of the set can be stored as any of the following: array, list, linked list, queue, stack, bit-array, hash table, balanced tree, binary search tree, etc.

Implementation using Predicates: the common property of set members can be implemented as a characteristic function, accessor function, or regular function.

Algebraic Specification of Sets

This algebraic specification is also a functional implementation (i.e. code) in a programming language designed for formal verification. The language (called ASL for Algebraic Specification Language) is programmed by specifying the abstract domain theory for a datatype. All implementation decisions are made by the *compiler*. The engine which operationalizes the code is very much like Prolog, that is, it is a logic pattern-matching engine with patterns compiled from the input specifications.

New functions and capabilities are added by *extending* a theory, that is, by building another theory which uses the base theory definitions and axioms. Application code consists of mathematical formulas to be evaluated.

```
theory TRIVIAL is

      sorts Elt

endtheory TRIVIAL


module BASICSET [ELT :: TRIVIAL] is

      sorts       Set

      functions
            Phi, Universe :         Set

            {_}:                    Elt -> Set

            _ symmetric-diff _ :    Set, Set -> Set
                                    (assoc comm ident: 0)
            _ intersect _ :         Set, Set -> Set
                                    (assoc comm idem ident: Universe)

      variables
            S,S',S'':               Set
            Elt,Elt':               Elt

      axioms
            (S sym-diff S) = Phi

            {Elt} intersect {Elt'} = Phi  :-  not(Elt = Elt')

            S intersect Phi = Phi

            S intersect (S' sym-diff S'')
                  = (S intersect S') sym-diff (S intersect S'')

endmodule BASICSET
```

```
module SET [X :: TRIVIAL] using NAT, BASICSET[X] is

      functions
            _ union _ :             Set, Set -> Set

            _ - _ :                 Set, Set -> Set

            #_ :                    Set -> Nat

      predicates
            _ member _ :            Elt, Set

            _ subset _ :            Set, Set

            empty :                 Set

            _ not-member _ :        Elt, Set

      variables
            X:                      Elt
            S,S',S'':               Set

      axioms
            S union S' = ((S intersect S') sym-diff S) sym-diff S'

            S - S' = S intersect (S sym-diff S')

            empty S :- S = Phi

            X member S :- {X} union S = S

            X not-member S :- {X} intersect S = Phi

            S subset S' :- S union S' = S'

            # Phi = 0

            #({X} sym-diff S) = #(S) - 1 :- X member S

            #({X} sym-diff S) = #(S) + 1 :- X not-member S


endmodule SET
```

Data Structures for Sets

The appropriate data structure to implement a mathematical object such as sets depends upon *resources*, *task,* and *context*. Things that effect the performance of set data structures include:

- size of elements
- complexity of elements (i.e. nested and hierarchical forms)
- size of the Universe of Discourse
- typical size of sets during an application
- maximum size of sets during an application
- frequency of use of each type of transformation
- available resources (processing power and memory size)

The choice of a data structures depends upon an *analysis of the task* it is to be used for. To implement a common mathematical object (such as sets, integers, strings, trees and logic), we build a model of the task which tells us the frequency of transforms and the types of elementary objects. We then match these characteristics against the known performance of the several implementation choices. When the task is unknown or likely to change (almost always the case), *defensive programming* can lower maintenance and modification costs through the use of more generic data structures. Naturally, data types that are built into a language have had their implementation decisions made independent of the task.

Implementing Sets

We need a representation which may require adding and deleting members, identifying membership, and performing basic set operations such as union and intersection.

doubly-linked lists
good for small sets
O(NlnN) operations

balanced trees
good for sorted lists with lots of look-up and for testing membership
O(NlnN) operations, membership in O(lnN)

hash tables
must be invertable for set enumeration
converts set to range of integers, easy for lookup
good for adding and deleting members

bit vectors
union and intersection are bit level
poor for enumerating elements
good for transforming sparse sets, bad for very large sets
O(n) operations

Hybrid representations, such as a linked segment list, can be constructed which optimize particular performance characteristics.