BASICS

Set Interpretation

```
sig S extends E {
      F: one T
  }

  fact {
      all s:S | s.F in T
  }
```

- s is a set
- s is a subset of E
- F is a relation which maps each s to exactly one T
- s is an element of S
- s.F composes the unary relation s with the binary relation F, returning the a unary realtion of type T

`sig FSObject {parent: lone Dir}` : set FSObject, relation parent which relates FSObjects to DIrs, at most one parent dir for each FSObejct.

Sig Declaration

```
sig A,B extends C // A and B are disjoint
sig A, B in C // not necessarily disjoint
one sig A // enforces that has to always be exactly one instance of that element
```

Fact Declaration

```
fact {File + Dir = FSObject} // all system objects are either files or directories
fact {
    FSObject in Root.*contents // the set of all fiule sysdtem objects is a subset of everything reachable from the Root by following the contents relation zero or more times
  }
```

Assert Declaration

```
  assert acyclic {
     no d: Dir | d in d.^contents
   }
   //the contents path is acyclic
```

=> an assert claims that something must be true due to the behaviour of the model

=> checked using `check` keyword, where Alloy either finds a counter example to the assertion or doesn't

Ternary Relations

```
  // A File System
   sig FileSystem {
     root: Dir,
     live: set FSObject,
     contents: Dir lone-> FSObject,
     parent: FSObject ->lone Dir
   }
   {
   // root has no parent
   no root.parent
   // live objects are reachable from the root
   live in root.*contents
   // parent is the inverse of contents
   parent = ~contents
}

contents in live -> live
// contents is in the set of the relation from live objects to live
objects
```

- Each file system is realted to exactly one directory, the root.
- the live relation rlates each file system to the set of file system objects it contains
- the set keyword allows the contents relation to relate FileSystem to any number of file system objects.
  => without means mapping only one-to-one !!
- the contents relation maps each file system to a binary relation from directories to file system objects.
- parents relation relates each file system to file system objects to directories

Run

```
  run example for exactly 1 FileSystem, 4 FSObject
  //alloy will try to find a solution to the model in which there is
exactly 1 Filesystem and 4 FileSystemObjects
```

ADVANCED

```
  root: Dir & live, //intersection
  parent: (live - root) ->one (Dir & live),
```

=> the root of a FileSystem is in the set of live objects of its FileSystem and not just any directory.

=> the parent relation maps every live object except the root to exactly one live Dir.

Alloy Model

```
1. pred move [fs, fs': FileSystem, x: FSObject, d: Dir] {
2.     (x + d) in fs.live
3.     fs'.parent = fs.parent - x->(x.(fs.parent)) + x->d
4. }
```

=> the predicate move is true if file system fs' is the result of moving file system object x to directory d in file system fs.

- Line 2: the object to be moved and the destination directory of the move must both exist in the prestate file system.
- Line 3: the parent relation in the post state is the same as the prestate except the mapping from x to x's old parent is replaced by the mapping from x to x's nwew parent d.

```
1. pred removeAll [fs, fs': FileSystem, x: FSObject] {
2.   x in (fs.live - fs.root)
3.   let subtree = x.*(fs.contents) | fs'.parent = fs.parent - subtree->
(subtree.(fs.parent))
4. }
```

- 2: the file system object to be deleted, x , must b in the prestate file system, fs, but that it cannot be the root of fs.
- 3: a let statement acts as a macro replacing the right side of the assignement whenever the left side of the assignement appears.

-

Advanced Checking

```
moveOkay: check {
   all fs, fs': FileSystem, x: FSObject, d:Dir |
     move[fs, fs', x, d] => fs'.live = fs.live
 } for 5
```

=> claims that the move operation does not alter the set of objects in the file system