Lists

```
module lists.

type id        list A -> list A -> o.
type memb      A -> list A -> o.
type append    list A -> list A -> list A -> o.

id nil nil.
id (X::L) (X::K) :- id L K.

memb X (X::L).
memb X (Y::L) :- memb X L.

append nil K K.
append (X::L) K (X::M) :- append L K M.

type  memb_and_rest    A -> list A -> list A -> o.

memb_and_rest X (X::L) L.
memb_and_rest X (Y::K) (Y::L) :- memb_and_rest X K L.
```

Path in a graph

```
module path.

kind node  type.
type adj   node -> node -> o.
type path  node -> node -> o.

type a, b, c, d    node.
type e     node.
type f     node.

adj a b.
adj a c.
adj b c.
adj c d.
adj e d.

path X Y :- adj X Y.
path X Z :- adj X Y, path Y Z.
```

Sorting

```
module sorting.
import lists.

type permutation  list A -> list A -> o.
type inorder      list int -> o.
type psort        list int -> list int -> o.

permutation nil nil.
permutation L (H::T) :-
  append V (H::U) L, append V U W, permutation W T.

inorder nil.
inorder (A::nil).
inorder (A::B::Rest) :- A =< B, inorder(B::Rest).

psort L1 L2 :- permutation L1 L2, inorder L2.
```

```
type split   int -> list int -> list int -> list int -> o.
type qsort   list int -> list int -> o.

split X nil nil nil.
split X (A::Rest) (A::S) B :- A =< X, split X Rest S B.
split X (A::Rest) S (A::B) :- A > X,  split X Rest S B.

qsort nil nil.
qsort (First::Rest) Sorted :-
  split First Rest Small Big,
  qsort Small SmallSorted,
  qsort Big BigSorted,
  append SmallSorted (First::BigSorted) Sorted.


type bsort        list int -> list int -> o.

bsort L1 L2 :-
  append Sorted (Big::Small::Rest) L1,
  Big > Small, !,
  append Sorted (Small::Big::Rest) BetterL1,
  bsort BetterL1 L2.
bsort L1 L1.
```