1995-1-3

```
fun elem (x,[]) = false
  | elem (x,y::ys) = (x = y) orelse elem (x,ys);

fun intersect (_,[]) = []
  | intersect ([],_) = []
  | intersect (x::xs,ys) = if elem (x,ys)
        then x::intersect (xs,ys)
        else intersect (xs,ys);

fun subtract (xs,[]) = xs
  | subtract ([],_) = []
  | subtract (x::xs,ys) = if elem (x,ys)
        then subtract (xs,ys)
        else x::subtract (xs,ys);

fun append ([],ys) = ys
  | append (x::xs,ys) = x::append (xs,ys);

fun union (xs,ys) = append (xs,subtract (ys,xs));

(* union : ''a list * ''a list -> ''a list *)
```

1998-1-1

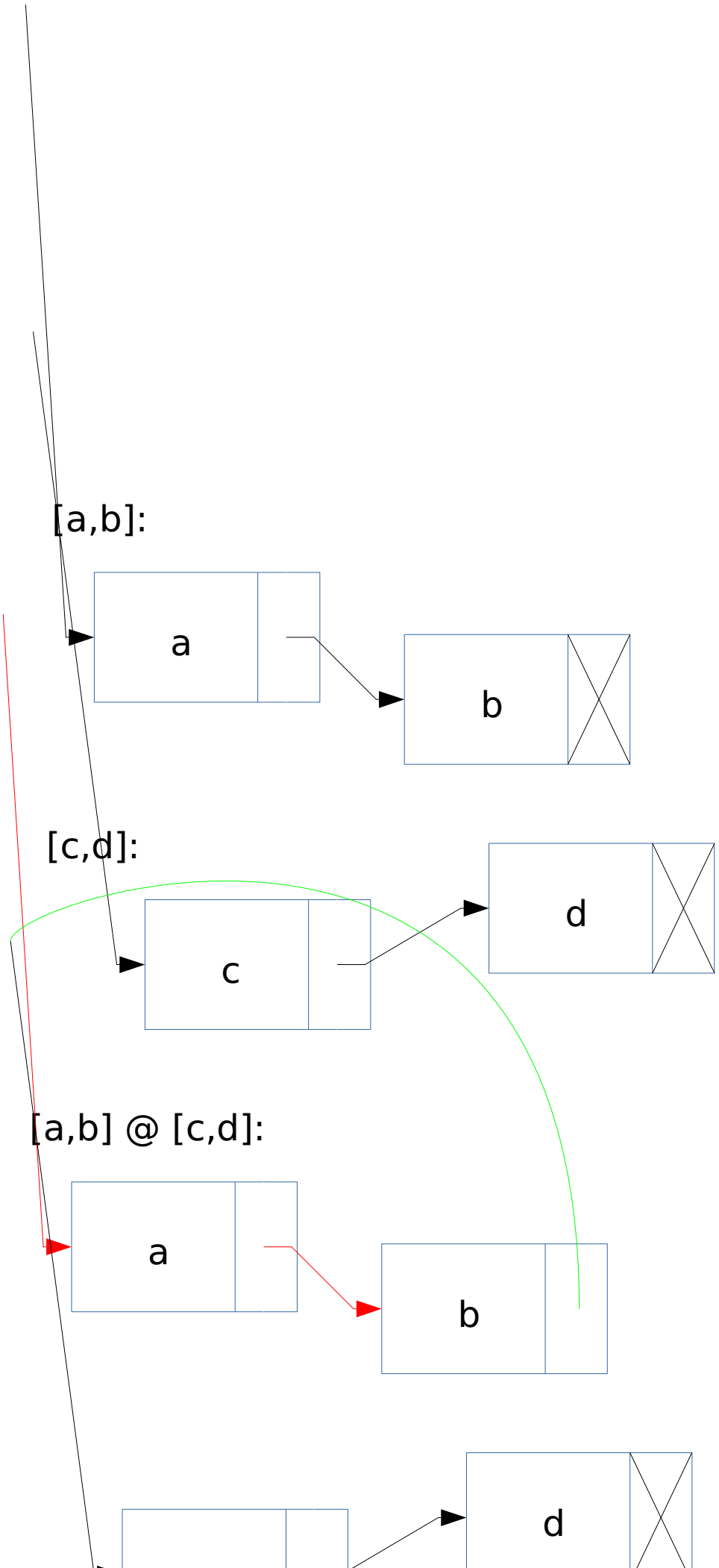
```
fun append ([],ys) = ys
  | append (x::xs,ys) = x::append (xs,ys);

fun partitioni acc [] = acc
  | partitioni (rs,x,ss) (y::ys) = if y < (x : real)
        then partitioni (y::rs,x,ss) ys
        else partitioni (rs,x,y::ss) ys;
fun partition (x::xs) = partitioni ([],x,[]) xs;

fun least (n,xs) = let
            val (small,mid,big) = partition xs
            val len = length small;
        in
            if n <= len
                then least (n,small)
                else if n = len + 1
                    then append (small,[mid])
                    else append (small,mid::least (n - len - 1,big))
        end;
```

2000-1-6

List elements are stored as the piece of data with a pointer to the next element in the list. There is also a pointer for the first item in the list. The last item in the list has a null pointer.


[a,b]:
a
b
[c,d]:
c
d
[a,b] @ [c,d]:
a
b
d

In computing [a,b] @ [c,d], the pointers of the first list are traversed until the null pointer is found. The null pointer is changed to the pointer for the second list. The efficiency of the append operation is not dependent on the length of the second list, so as l grows, the efficiency will not change.

1995-1-6

```
datatype bit = one | zero;
datatype cardinal = Cardinal of bit list;
exception Result_would_be_negative;

fun bnot zero = one
  | bnot one = zero;
fun bxor zero zero = zero
  | bxor zero one = one
  | bxor one zero = one
  | bxor one one = zero;
fun biff x y = bnot (bxor x y);
fun band one one = one
  | band _ _ = zero;
fun bor zero zero = zero
  | bor _ _ = one;

fun succ (zero::xs) = one::xs
  | succ (one::xs) = zero::succ xs
  | succ [] = [one];
fun pred (one::xs) = zero::xs
  | pred (zero::xs) = one::pred xs
  | pred [] = raise Result_would_be_negative;

fun addCarry zero (x::xs) (y::ys) = bxor x y::addCarry (band x y) xs ys
  | addCarry one (x::xs) (y::ys) = biff x y::addCarry (bor x y) xs ys
  | addCarry zero [] ys = ys
  | addCarry zero xs [] = xs
  | addCarry one [] ys = succ ys
  | addCarry one xs [] = succ xs;
fun add (Cardinal xs) (Cardinal ys) = Cardinal (addCarry zero xs ys);

fun subtractInner (one::xs) (y::ys) = bnot y::subtractInner xs ys
  | subtractInner (zero::xs) (zero::ys) = zero::subtractInner xs ys
  | subtractInner (zero::xs) (one::ys) = one::subtractInner (pred xs) ys
  | subtractInner xs [] = xs
  | subtractInner [] _ = raise Result_would_be_negative;
fun subtract (Cardinal xs) (Cardinal ys) = Cardinal (subtractInner xs ys);

fun multiplyInner (one::xs) ys = addCarry zero ys (zero::multiplyInner xs ys)
  | multiplyInner (zero::xs) ys = zero::multiplyInner xs ys
  | multiplyInner [] _ = [];
fun multiply (Cardinal xs) (Cardinal ys) = Cardinal (multiplyInner xs ys);
```