

```
fun append (xs,ys) =  
  if xs=[]  
  then ys  
  else (hd xs)::append(tl xs,ys)  
  
fun map (f,xs) =  
  case xs of  
    [] => []  
  | x::xs' => (f x)::(map(f,xs'))  
  
val a = map (increment, [4,8,12,16])  
val b = map (hd, [[8,6],[7,5],[3,0,9]])
```

Programming Languages

Dan Grossman

Nested Patterns Precisely

(Most of) the full definition

The **semantics** for pattern-matching takes a pattern p and a value v and decides (1) does it match and (2) if so, what variable bindings are introduced.

Since patterns can nest, the **definition is elegantly recursive**, with a separate rule for each kind of pattern. Some of the rules:

- If p is a variable x , the match succeeds and x is bound to v
- If p is $_$, the match succeeds and no bindings are introduced
- If p is $(p1, \dots, pn)$ and v is $(v1, \dots, vn)$, the match succeeds if and only if $p1$ matches $v1$, ..., pn matches vn . The bindings are the union of all bindings from the submatches
- If p is $C\ p1$, the match succeeds if v is $C\ v1$ (i.e., the same constructor) and $p1$ matches $v1$. The bindings are the bindings from the submatch.
- ... (there are several other similar forms of patterns)

Examples

- Pattern `a :: b :: c :: d` matches all lists with ≥ 3 elements
- Pattern `a :: b :: c :: []` matches all lists with 3 elements
- Pattern `((a, b), (c, d)) :: e` matches all non-empty lists of pairs of pairs