13 Church Encoding

= ”fold“ over the respective represented object (number, list, …)

13 Church Encoding

= ”fold“ (catamorphism) over the respective represented object (number, list, …)

know: fold over lists

```
def fold(l)(f,e) = match l with
  []   => e
  h::t => f(h, fold(t)(f,e))
```

13 Church Encoding

```
def fold(l)(f,e) = match l with
  []   => e
  h::t => f(h, fold(t)(f,e))

// Church encoding of [1,2,3]
def ce_123 : (A->B—>B,B)->B =
  fold([1,2,3])
```

13 Church Encoding

```
def fold(l)(f,e) = match l with
  []    => e
  h::t => f(h, fold(t)(f,e))
```

lists can be specialized to Peano nat.s:

```
// Nat = List of Unit
def fold(n)(f,e) = match n with
  []     => e
  ()::t => f((), fold(t)(f,e))
```

13 Church Encoding

```
def fold(l)(f,e) = match l with
   []   => e
   h::t => f(h, fold(t)(f,e))
```

lists can be specialized to Peano nat.s:

```
// Nat = List of Unit
def fold(n)(f,e) = match n with
   []    => e
   ()::t => f((), fold(t)(f,e))
```

always unit, no extra information at each position

13 Church Encoding

```
// Nat = List of Unit
def fold(n)(f,e) = match n with
  []     => e
  ()::t => f((), fold(t)(f,e))
```

compare:

```
def fold(n)(f,e) = match n with
  0     => e
  S(m) => f(fold(m)(f,e))
```

13 Church Encoding

```
def fold(n)(f,e) = match n with
   0     => e
   S(m)  => f(fold(m)(f,e))
```

In other words:
fold over **n** = iterate **n** times the given function f starting with the given number e

```
f(f(…f(e)…))
```

13 Church Encoding

```
f(f(…f(e)…))
```

compare lists:

```
f((),f((),…f((),e)…))
f(1 ,f(2 ,…f(n ,e)…))
```

13 Church Encoding

approach generalizes to other structures:
trees, booleans, …