Time and change

- In the functional paradigm, there is no notion of time
  - All functions are mathematical functions; once defined they never change
  - Programs do execute on a real machine, but a program cannot observe the execution of another program or of part of itself
    - It can only see the results of a function call, not the execution itself
    - Observing an execution of a program can only be done outside of the program's implementation
- In the real world, there is time and change
  - Organisms change their behavior over time, they grow and learn
  - How can we model this in a program?
- We need to add time to a program
  - Time is a complicated concept! Let us start with a simplified version of time, an abstract time, that keeps the essential property that we need: modeling change.

State as an abstract time (1)

- Here's one solution: We define the abstract time as a sequence of values and we call it a state
- A state is a sequence of values calculated progressively, which contains the intermediate results of a computation
- The functional paradigm can use state according to this definition!
- The definition of Sum given here has a state

```
fun {Sum Xs A}
   case Xs
   of nil then A
   [] X|Xr then
         {Sum Xr A+X}
   end
end


{Browse {Sum [1 2 3 4] 0}}
```

State as an abstract time (2)

- The two arguments Xs and A give us an implicit state

Xs	A
[1 2 3 4]	0
[2 3 4]	1
[3 4]	3
[4]	6
nil	10

- It is implicit because the language has not changed
  - It is purely in the programmer's head: the programmer observes the changes in the program
- In most cases this is not good enough: we want the program itself to observe the changes
  - We need a language extension!
  - *We leave the functional paradigm and enter another paradigm*

```
fun {Sum Xs A}
   case Xs
   of nil then A
   [] X|Xr then
         {Sum Xr A+X}
   end
end


{Browse {Sum [1 2 3 4] 0}}
```