Introduktion til Programmering og Problemløsning (PoP)

Design af rekursive funktioner

Jon Sporring
Department of Computer Science
2022/9/28

UNIVERSITY OF COPENHAGEN

Rekurssionsregler

1. Funktionen skal kalde sigselv evt. indirekte
2. Der skal være en stopbetingelse
3. Stopbetingelsen skal nås på et tidspunkt

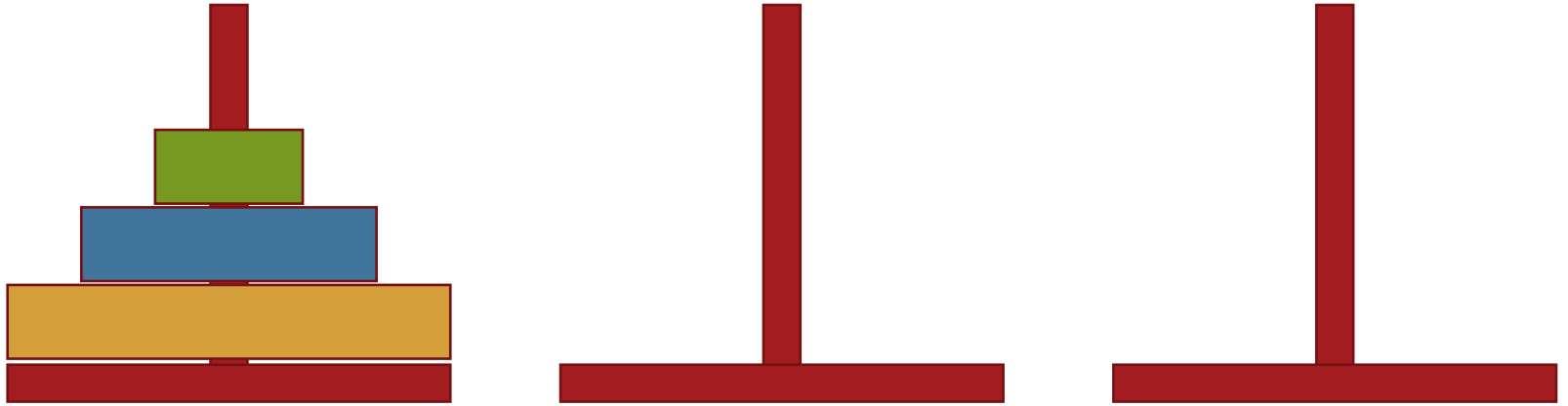
```
let rec sum (n: int) : int =
  if n = 0 then 0
  else n + sum (n - 1);;
```

```
let rec fib (n: int) : int =
  if n < 1 then 0
  elif n = 1 then 1
  else fib (n - 1) + fib (n - 2);;
```

```
let rec fac (n: int) : int =
  if n = 0 then 1
  else n * fac (n - 1);;
```

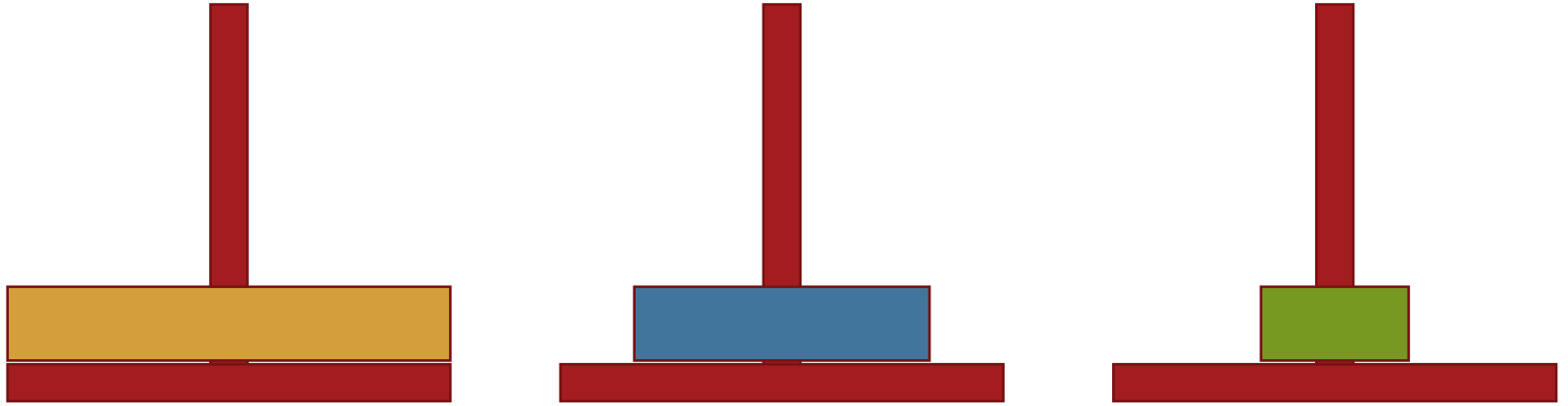
```
let rec includes (elm: 'a) (lst: 'a list): bool =
  match lst with
    [] -> false
    | v::rst ->
      if v = elm then true
      else includes elm rst
```

Hanoitårnene

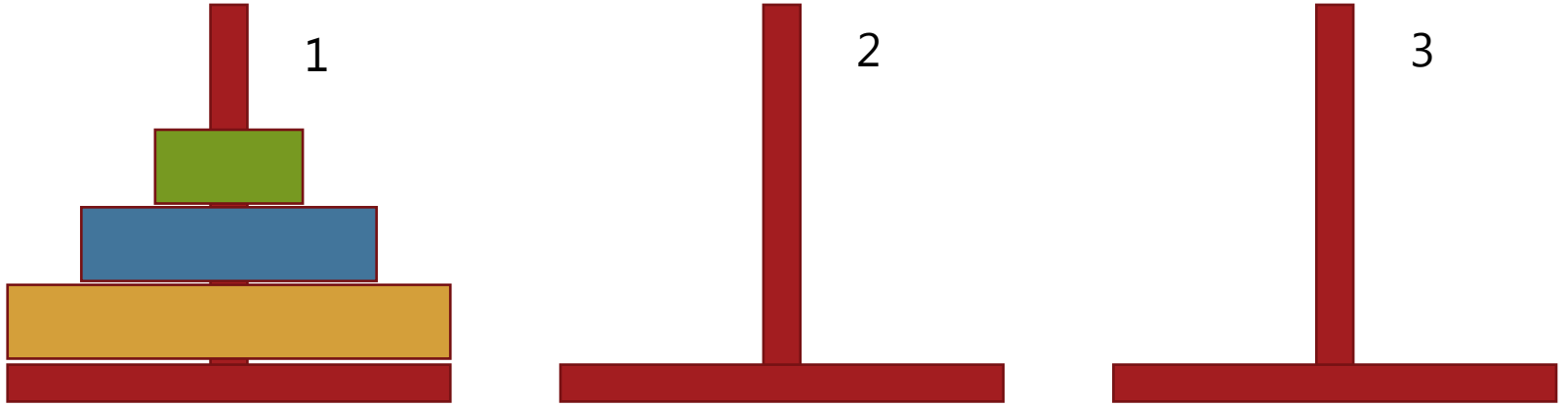


Spilleregler:

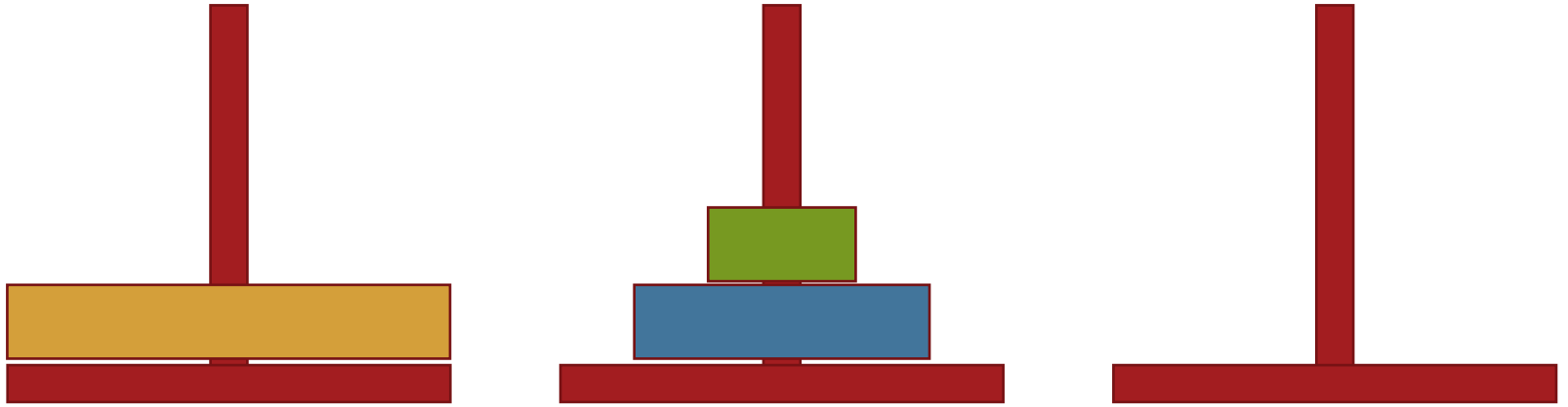
1. Spillet spilles med N skiver, der kan placeres på tre pinde.
2. Udgangspunktet er at alle skiverne ligger i orden på den første pind.
3. Spillet spilles ved at flytte skiverne en af gangen, således at alle skiver ender på den fjerneste pind.
4. På intet tidspunkt må en stor skive ligge ovenpå en mindre skive.



Hanoitårnene



Problem: Kan vi finde et rekursionsskridt (som ikke bryder reglerne)?
Observation: Et delmål er at få gul brik på 3. pind. Dvs. blå og grøn skal på 2. pind.

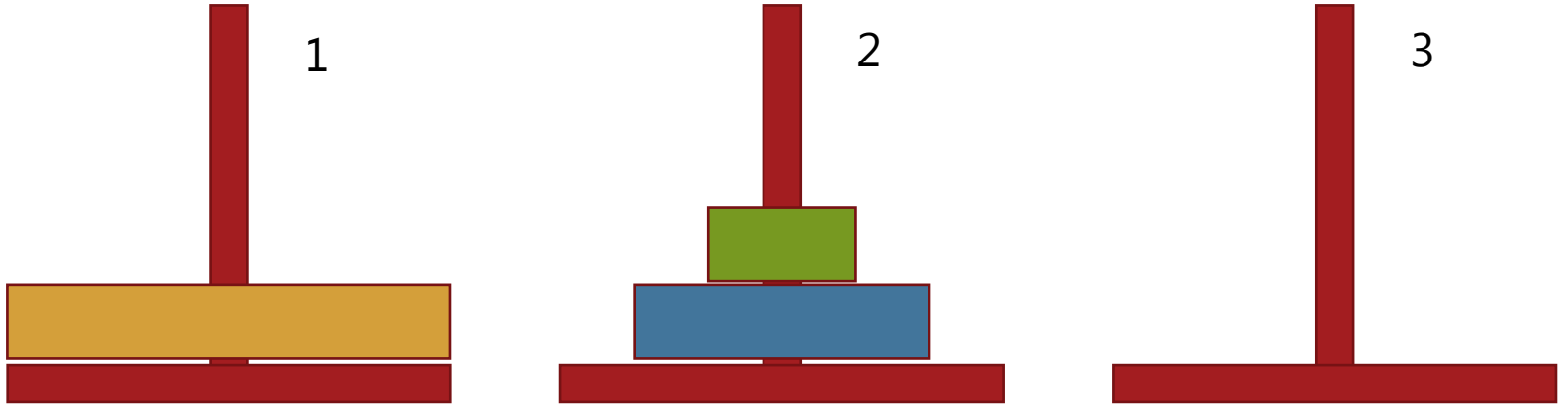


Skitse: For at flytte 3 brikker fra pind 1 til pind 3 skal vi

1. Flytte 2 øverste brikker fra pind 1 til 2
2. Flytte brik 3 til pind 3
3. Flytte 2 brikker fra pind 2 til 3

Rekursion

Hanoitårnene



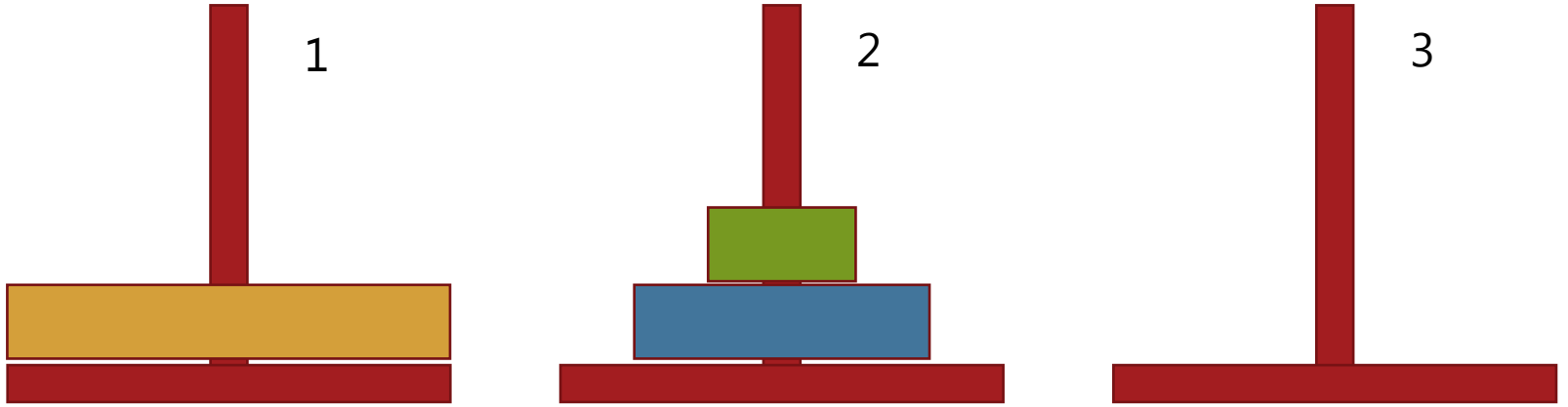
Skitse: For at flytte 3 brikker fra pind 1 til pind 3 skal vi

1. Flytte 2 øverste brikker fra pind 1 til 2
2. Flytte brik 3 til pind 3
3. Flytte 2 brikker fra pind 2 til 3

Rekursion

```
// hanoi: n: int -> src: int -> aux: int -> tgt: int -> board -> board
let brd: board = [[1;2;3];[];[]]
hanoi 3 0 1 2 brd // Move 3 pieces from peg 1 to 3 with the help of peg 2
```

Hanoitårnene



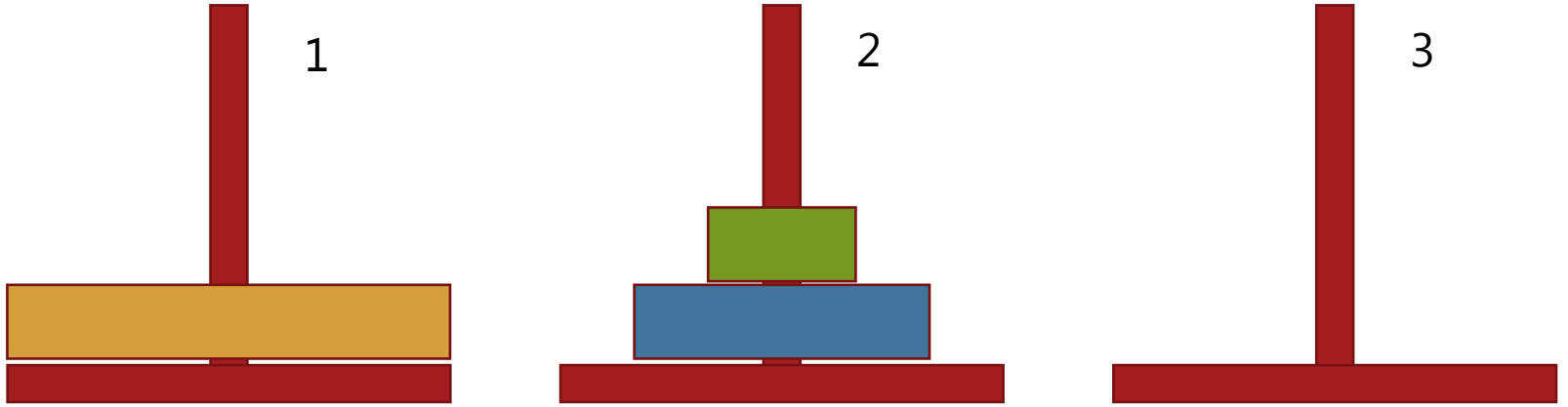
Skitse: For at flytte 3 brikker fra pind 1 til pind 3 skal vi

1. Flytte 2 øverste brikker fra pind 1 til 2
2. Flytte brik 3 til pind 3
3. Flytte 2 brikker fra pind 2 til 3

Rekursion

```
type piece = int
type peg = piece list
type board = peg list
// hanoi: n: int -> src: int -> aux: int -> tgt: int -> board -> board
let brd: board = [[1;2;3];[];[]]
hanoi 3 0 1 2 brd // Move 3 pieces from peg 1 to 3 with the help of peg 2
```

Hanoitårnene



Skitse: For at flytte 3 brikker fra pind 1 til pind 3 skal vi

1. Flytte 2 øverste brikker fra pind 1 til 2
2. Flytte brik 3 til pind 3
3. Flytte 2 brikker fra pind 2 til 3

Rekursion

```
type piece = int
type peg = piece list
type board = peg list
let rec Hanoi (n: int) (src: int) (aux: int) (tgt: int) (brd: board) : board =
  if n = 0 then brd
  else
    let topMoved = hanoi (n-1) src tgt aux brd
    let bottomMoved = move src tgt topMoved
    hanoi (n-1) aux src tgt bottomMoved
// hanoi: n: int -> src: int -> aux: int -> tgt: int -> board -> board
let brd: board = [[1;2;3];[];[]]
hanoi 3 0 1 2 brd // Move 3 pieces from peg 1 to 3 with the help of peg 2
```

Antag at det virker

Hanoitårnene

```
type piece = int
type peg = piece list
type board = peg list
let move (src: int) (tgt: int) (brd: board) =
  let pc = brd[src].Head
  let newSrc = brd[src].Tail
  let newTgt =  pc :: brd[tgt]
  let rec helper (n: int) (p: board) : board =
    match n with
      i when i >= 3 -> []
      | i when i = src -> newSrc :: helper (n+1) p.Tail
      | i when i = tgt -> newTgt :: helper (n+1) p.Tail
      | _ -> p.Head :: helper (n+1) p.Tail
  helper 0 brd
let rec Hanoi (n: int) (src: int) (aux: int) (tgt: int) (brd: board) : board =
  if n = 0 then brd
  else
    let topMoved = hanoi (n-1) src tgt aux brd
    let bottomMoved = move src tgt topMoved
    hanoi (n-1) aux src tgt bottomMoved
// hanoi: n: int -> src: int -> aux: int -> tgt: int -> board -> board
let brd: board = [[1;2;3];[];[]]
hanoi 3 0 1 2 brd // Move 3 pieces from peg 1 to 3 with the help of peg 2
```

Hanoitårnene

```
% dotnet fsi src/hanoi.fsx
[[1; 2; 3]; []; []]
[[2; 3]; []; [1]]
[[3]; [2]; [1]]
[[3]; [1; 2]; []]
[[]; [1; 2]; [3]]
[[1]; [2]; [3]]
[[1]; []; [2; 3]]
[[]; []; [1; 2; 3]]
```

hanoi.fsx

Users > jrh630 > repositories > PoP > lectures > 05Recursion > src > hanoi.fsx

```
1   type piece = int
2   type peg = piece list
3   type board = peg list
4
5   let move (src: int) (tgt: int) (brd: board) =
6     let pc = brd[src].Head
7     let newSrc = brd[src].Tail
8     let newTgt =  pc :: brd[tgt]
9     let rec helper (n: int) (p: board) : board =
10      match n with
11        i when i >= 3 -> []
12        | i when i = src -> newSrc :: helper (n+1) p.Tail
13        | i when i = tgt -> newTgt :: helper (n+1) p.Tail
14        | _ -> p.Head :: helper (n+1) p.Tail
15    helper 0 brd
16
17  let rec hanoi (n: int) (src: int) (aux: int) (tgt: int) (brd: board) =
18    if n = 0 then brd
19    else
20      let newBrd =
21        brd
22        |> hanoi (n-1) src tgt aux
23        |> move src tgt
24      printfn "%A" newBrd
25      hanoi (n-1) aux src tgt newBrd
26
27  // hanoi: n: int -> src: int -> aux: int -> tgt: int -> board -> board
28  let brd: board = [[1;2;3];[];[]]
29  printfn "%A" brd
30  hanoi 3 0 1 2 brd // Move 3 pieces from peg 1 to 3 with the help of peg 2
31
```

Restricted Mode ⊗ 0 ⚠ 0 Ln 31, Col 1 Spaces: 2 UTF-8 LF F#

piping

kommunikation

Resumé

- Rekursionsreglerne
- Lad som om det allerede er løst