Introduction to Functional Programming in *OCaml*

Roberto Di Cosmo, Yann Régis-Gianas, Ralf Treinen

Week 3 - Sequence 2: Tree-like values

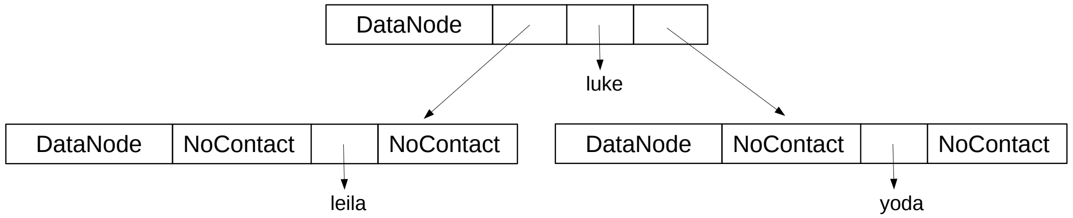
A tree-like representation for databases

- Consider the following **tree-like representation** for databases:

```
type database =
  | NoContact
  | DataNode of database * contact * database
```

- We will enforce an **invariant**.
- A database node `DataNode (left, c, right)` is well-formed if
  - every contact in `left` is lexicographically smaller than `c`;
  - every contact in `right` is lexicographically greater than `c`.

In the machine



- is the representation of

```
DataNode(DataNode(NoContact, leila, NoContact),
         luke,
         DataNode(NoContact, yoda, NoContact))
```

- This value fulfills our invariant!

Looking for a contact

```
let search db name =
  let rec traverse = function
    | NoContact ->
      Error
    | DataNode (left, contact, right) ->
      if contact.name = name then
        FoundContact contact
      else if name < contact.name then
        traverse left
      else
        traverse right
  in
  traverse db
```

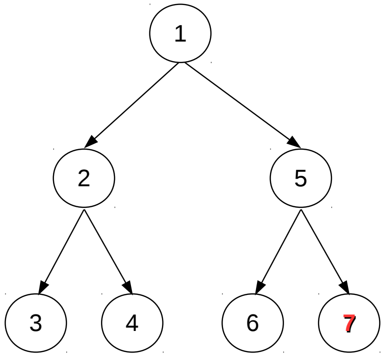
A more efficient lookup

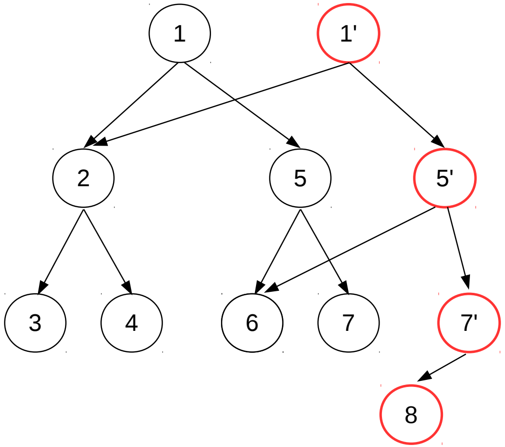
- In the worst case, the contact is not found and we have crossed a number of nodes which is bounded by the height of the tree.
- In the array-based implementation, the entire database is traversed.
- It is unlikely that the height of the tree is equal to the number of contacts! (This would mean that the tree is degenerated into a list.)
- As an exercise, try to maintain the extra invariant that the tree is balanced, i.e. that its height is bound by the logarithm of the number of contacts.

Inserting a contact

```
let insert db contact =
  let rec traverse t =
    match t with
      | NoContact ->
        DataNode (NoContact, contact, NoContact)
      | DataNode (left, contact', right) ->
        if contact.name = contact'.name then
          t
        else if contact.name < contact'.name then
          DataNode (traverse left, contact', right)
        else
          DataNode (left, contact', traverse right)
  in
  traverse db
```

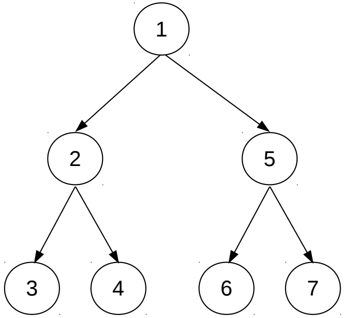
Insertion shares subtrees between databases





Removing an element

- Removing an element seems a bit complicated. . .
- We should be able to **focus on** the tree problem **independently of** the fact that it represents a database.
- This is the **separation of concerns** principle.



Forthcoming **parameterized types** will help us perform such a **modular development**.