**School of Computing and Information Systems**
**COMP30026 Models of Computation Tutorial Week 3**

7–11 August 2017

## Plan

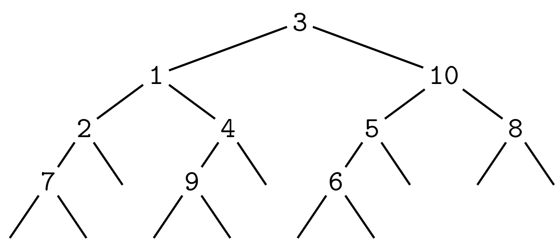
This week there are lots of exercises—many of them short and quick. Generally we will give you more exercises than we can cover in a tute; finish them off in your own time. Questions 6–9 are about Haskell; Question 9 is actually a bunch of exercises, to do on Grok. Most are about Haskell for tree manipulation. Binary trees are recursively defined structures, and the style of programming needed to solve Question 7 is important; we will need it frequently. Questions 10–14 are about propositional logic.

## The exercises

6. Consider a Haskell function `build_tree` which takes a list and creates a balanced binary tree out of the elements of the list. Given the list `[7,2,1,9,4,3,6,5,10,8]` it may yield

```
Node 3 (Node 1 (Node 2 (Node 7 Void Void) Void)
               (Node 4 (Node 9 Void Void) Void)
       )
       (Node 10 (Node 5 (Node 6 Void Void) Void)
                (Node 8 Void Void)
       )
```

We have laid the expression out so it can more easily be read. We can depict it graphically:



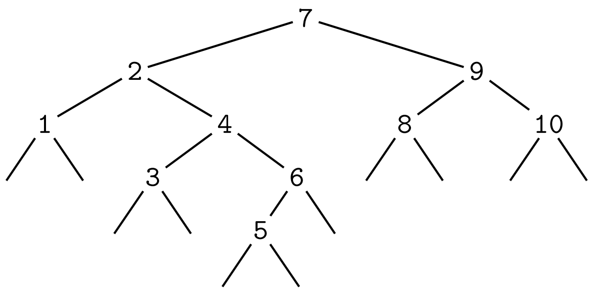

The code for `buildtree` may look as follows:

```
data BinTree a
  = Void | Node a (BinTree a) (BinTree a)
    deriving (Eq, Show)

buildtree :: [a] -> BinTree a
buildtree []
  = Void
buildtree xs
  = Node midval (buildtree as) (buildtree (tail bs))
    where
      halflength = length xs `div` 2
      midval     = xs !! halflength
      (as,bs)    = splitAt halflength xs
```

Which Haskell functions or features used here have you not met so far? Speculate and discuss what purpose they might serve. Discuss how you would write a function `contents` which takes a binary tree such as the one above and generates the elements as a list. Come on—do it; you will need it for the Grok worksheet for this week. See if you can make `contents` act as the inverse function to `buildtree`—the composition `contents . buildtree` should be the identity function on lists.

7. The Grok worksheet will also ask you to write a function `buildbst` which takes a list and creates a *binary search tree* out of the elements of the list. For the example list above, it may build this tree:




If you get it right then the composition `contents . buildbst` is a sorting function!

8. Beware of misconceptions about Haskell's list notation. What is the type of `f` defined below? Is it actually well-typed? Did somebody forget the square brackets in the last equation?

```
f []  = 0
f [x] = x
f y   = 42
```

9. Find one or two fellow students and work through the questions on this week's Grok worksheet.

10. For each of the following pairs, indicate whether the two formulas have the same truth table.

(a) $\neg P \Rightarrow Q$ and $P \Rightarrow \neg Q$
(b) $\neg P \Rightarrow Q$ and $Q \Rightarrow \neg P$
(c) $\neg P \Rightarrow Q$ and $\neg Q \Rightarrow P$
(d) $P \Rightarrow (Q \Rightarrow R)$ and $Q \Rightarrow (P \Rightarrow R)$
(e) $P \Rightarrow (Q \Rightarrow R)$ and $(P \Rightarrow Q) \Rightarrow R$
(f) $(P \Rightarrow Q) \Rightarrow P$ and $P$
(g) $P \vee Q \Rightarrow R$ and $(P \Rightarrow R) \wedge (Q \Rightarrow R)$

11. Find a formula that is equivalent to $P \Leftrightarrow (P \wedge Q)$ but simpler, that is, using fewer symbols.

12. Recall that $\oplus$ is the "exclusive or" connective. Show that $(P \oplus Q) \oplus Q$ is equivalent to $P$.

13. Recall that $\Leftrightarrow$ is the biimplication connective. Show that $(P \Leftrightarrow Q) \equiv (\neg P \Leftrightarrow \neg Q)$.

14. Show that $P \Leftrightarrow (Q \Leftrightarrow R) \equiv (P \Leftrightarrow Q) \Leftrightarrow R$. This tells us that we could instead write

$$P \Leftrightarrow Q \Leftrightarrow R \tag{1}$$

without introducing any ambiguity. Mind you, that may not be such a good idea, because many people (incorrectly) tend to read "$P \Leftrightarrow Q \Leftrightarrow R$" as

$$P, Q, \text{ and } R \text{ all have the same truth value} \tag{2}$$

Show that (1) and (2) are incomparable, that is, neither is a logical consequence of the other.