COMP 523: Language-based security
Assignment 3 (100 points total)

Prof. B. Pientka
McGill University

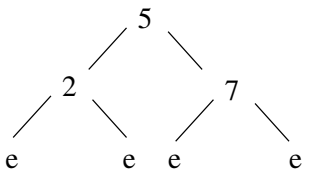
September 22, 2010—**Due: Wednesday, 29 September 2010 at 2:35pm**

Exercise 1 (15 points) In this question we explore balanced binary trees.

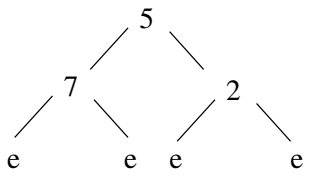
5 points Implement a data-type `bal_tree` where we index the binary tree with its height. Note that since it is a completely balanced binary tree, both its children must have the same height. For simplicity, we only will store natural numbers at each node but nothing at the leafs.

10 points Implement the function `mirror` which accepts a balanced binary tree and switches each of its subtree. So the left child becomes the right child and the right child becomes the left child.

Balanced Binary Tree



Mirrored Balanced Binary Tree



Fill in your code in the file `baltree.bel`.

Exercise 2 (20 points): In class, we have proven that if `leq M N` then `leq M (succ N)`. We recall the definition of `leq` and give an inductive definition of equality for natural numbers below.

```
leq: nat → nat → type .
le_z : leq z N.
le_s : leq N M → leq (succ N) (succ M).

eq : nat → nat → type .
eq_z : eq z z.
eq_s : eq N M → eq (succ N) (succ M).
```

Implement the proof for the following theorem as a recursive function: If `eq M N` then `leq M N`.

Fill in your code in the file `eq-proof.bel`

Exercise 3 (35 points) : We explore some simple meta-theoretic properties of the language and operational semantics we have seen in class.

20 points Implement a function `vsound: (eval M V)[ ]` $\rightarrow$ `(value V)[ ]` which corresponds to the value soundness theorem which can be stated as follows:

Theorem 0.1 (Value soundness) *If* `eval M V` *then* `value V`.

15 points Implement the function `vself : (value V)[ ]` $\rightarrow$ `(eval V V)[ ]` which corresponds to the theorem that values evaluate to themselves.

Theorem 0.2 (Values evaluate to themselves) *If* `value V` *then* `eval V V`.

Fill in your code in the file `vsound.bel`

Exercise 4 (25 points): In this question, we explore a certifying type-preserving evaluator. Modify the big-step evaluator from the last homeowork to make it certifying, i.e. it will not simply return the final value but also a derivation how such a value can be derived.

Fill in your code in the file `cert-eval.bel`.