# Homework 28

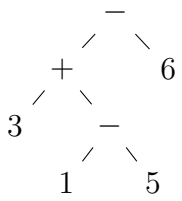
Robbie McKinstry, Jack McQuown, Cyrus Ramavarapu

7 November 2016

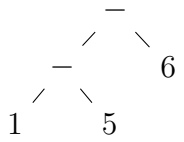
## Problem 23:

The parallel evaluation of an expression tree containing integers at the leaves and the four primary arithmetic operations at the internal nodes can be performed in a manner very similar to the parallel evaluation of a similar expression tree with only addition and subtraction as the key operations. To simplify the following argument, these two problems will be referred to as *4-ExTree* and *2-ExTree*.

When solving *2-ExTree*, the original expression tree is generalized through a cutting process of the internal nodes in order to transform the symbolic operation on the node to the evaluation of a function on the edges. For example, the processor on the "+" sign in the second level would perform the following transformation:




and produce:




This process can be repeated for every other node, since this problem is being solved on an CREW PRAM. However, the difficulty arises when the nodes must be partitioned into two halves. This can be done by an *Eulerian Tour* over the tree after each cut by labeling the leaves either a 0 or a 1.

After each cutting process the number of evaluations that need to be computed will go down by $1/8^{th}$ of the original problem size. Together with the renumbering, this gives a runtime for *2-ExTree* of $\mathcal{O}\big(\log^2 n\big)$.

Generalizing this problem to an expression tree also containing multiplication and division requires expanding the possible functional evaluations made on the edges to include new operations. The only difference that this problem has with *2-ExTree* is that the expressions will now include "$\times$" and "$\div$". Since the same algorithm can be still used, the runtime will remain $\mathcal{O}\big(\log^2 n\big)$.

The case where division by zero occurs is one of language semantics. This behavior is naturally undefined in an abstract machine; any particular semantic interpretation fails to translate into concrete interpretations. Because this form of the problem is undefined, the behavior is likewise undefined in our algorithm. Any interpretation is outside of the scope of this problem because it's semantics specific.

# Problem 24:

The solution this problem is the same as the above problem. Just as addition, subtraction multiplication, and division are closed over the Real numbers, AND, NOT, and OR are closed over binary numbers. To solve a bitwise form of the binary expression tree evaluation problem, simple replace the arithmetic function compositions with logical function compositions.