Proof-COL226

Navneel Mandal
2017CS50414

To prove that the implementation using eval and compile-stackmc give the same answer.

I would be breaking down the proof into 3 parts, with first 2 to show the validity of the answer of the compile and stackmc functions and the 3rd part showing equality.

1) **Proving that the compile function returns the postorder list of an exptree.**

Proof: Induction on height(tree).

Base Case: When height(tree) = 0 i.e The tree only has a leaf.
In such a scenario, the tree can only have N as the datatype of the leaf as any other operation requires the tree to have a height of atleast 1. The compile function returns CONST y as the only element in the list where y is the bigint of the int dataype in exptree. This is desirable, since the postorder of a single leaf element is the leaf itself. So, the Base Case is satisfied.

Induction Hypothesis: Suppose we have shown for all trees having height(tree) <= k:
compile(tree) = postorder(tree)

Induction Case(s): For a tree having height = k+1:
Let the subtrees of this tree be t1 and t2, left and right respectively.
The root will have the height k+1, but t1 will have height k and t2 will have height <= k or vice versa. Since we know compile function gives the post order traversal of trees having height <= k, we can now say that:

compile(t1) = postorder(t1)
and, compile(t2) = postorder(t2)

Visualising the whole tree now as the root node and 2 subtrees t1 and t2, compile function will now give:

compile(tree) = compile(t1) @ compile (t2) @ compile(root)
i.e compile(tree) = postorder(t1) @ postorder(t2) @ [root]
compile(tree) = postorder(tree)

2) Proving that stackmc function returns a bigint for a correctly formed opcode list.

Proof: Induction on length(opcode list).

Base Case: When the list has only 1 element. In this case, the list can have the only element in the form of CONST y where y is a bigint. Now the stackmc will put this element in the originally empty bigint list and pop it from the opcode list and call the stackmc function recursively with updated parameters. Since now the opcode list is empty, the fuction returns the head of the bigint list which is a bigint.

Induction Hypothesis: Suppose we have shown for all opcode lists of length <= k:
stackmc [] oplist returns a bigint.

Induction Cases: For a list of length k+1, the last element can be
(i) A binary operator
(ii) A unary operator

It can’t be a CONST. Since that would mean the exptree from which this OPCODE list was made had a (big)int as a non-leaf which would violate the condition of exptree.

If the last element is a binary operator, then the list can be broken down into three parts each having length < k and returning a bigint while the last part is the Binary Operator itself.
oplist = part 1+ part 2+ Binary Operator
stackmc [] oplist = stackmc [bigint1; bigint2] [Binary Operator]

bigint1 and bigint2 will be present in the bigintlist, and on encountering the binary operator it’ll pop the 2 elements for the operation by the binary operator and return a bigint in the process.

If the last element is a unary operator, it means that the k part of the list will return a bigint on which the unary operator will apply the last operation. So, even it’ll return a bigint.

Proving that eval and compile-stackmc returns the same answer.

Proof: Induction on height of the exptree.

Base Case: height(tree) = 0 i.e only a leaf of the tree exists. In this case,eval returns the int itself since the leaf can only be a (big)int. Now, the compile function will return the OPCODELIST having one element of the form CONST y where y is a bigint. And the stackmc function will pop this CONST from the opcode list and push it to the bigint list which then returns it. So, since both function return the same number, they are equal, only the datatypes are different.

Induction Hypothesis: Suppose we have shown for all trees having height <= k that the answer returned by eval and compile-stackmc is the same.

Induction Case(s): For a tree having height k+1, we can visualise this tree having subtrees T1 with height k and T2 with height <= k or vice versa.

We know that for subtrees T1 and T2, the asnwer by eval and compile-stackmc is the same, so let the answer for T1 be ans1 and for T2 is ans2. ans1 and ans2 can be bigint or int, we are only concerned with the value in this proof.

We can now consider it a fresh problem with an exptree having the root, ans1 and ans2. The height of this tree will be 1, and since 1 <= k , the eval and the compile-stackmc will return the same answer.

So, we have completed the proof that eval and compile-stackmc return the same answer barring the datatypes.