Assignment 1- Paper Exercise

As a paper exercise, you have to state and prove the theorem that your compiler+stack machine *calculates* a bigint whose value is the same as what eval *computes*.

Claim, P(h): Given any exptree, t, of height h, (eval t) = stackmc stk (compile t)

Assumptions: 1. Bigint operations and normal ocaml inbuilt operations give the same result.
2. Bigint and int differ in only representations, for the purpose of this proof/ assignment, and they are equated freely without loss of implications.

We shall perform induction over h to support our claim.

Base step,P(0):

Consider an exptree t, of height 0. The tree can only be of form N of int, say N(a).

(Eval t) will match t to N of some constant a, and return the constant a.

(Compile t) will also match t to N of some constant a, and return a list with only 1 element : CONST of (bigint form of) constant a. Call this list pgm
stackmc stk pgm will, in the first iteration, push a to the stack and recursively call itself by passing the modified stack with the tail of pgm, which is, an empty list.
In this iteration, seeing that pgm is empty, stackmc returns the top value of stk, that is a.

Therefore the base case holds true.

We now assume all P(i) to be true for i less than equal to some integer k, for the sake of the our inductive proof.

To show: P(k+1) is true

Consider an exptree t, of height k+1. 2 cases arise.

Case 1: t is of the form <oper> of exptree, where <oper> is a unary operation.
Let t' be the child of t.

Eval t will equate to <oper> (eval t')

(Compile t) will operate to give a conjunction of lists, (compile t') and [<oper>]. Let this list be of length l.

Passing the above to stackmc as pgm, with any arbitrary list as stk, we observe that stackmc will continue modifying stk as per pgm and recursively call itself, each time, shortening the length of pgm by 1. By the time only 1 element is left in pgm, we notice that the top most element of stk will be equivalent to (stackmc stk (compile t')), as the first l-1 elements in pgm are identical to (compile t'). Note that t' is an exptree with height less than or equal to k (as the height of its parent, t, is itself k+1), and therefore by our inductive assumption, (stackmc stk (compile t')) is equal to (eval t'). Therefore, by the time pgm becomes of length 1, the top element of stk will be equal to (eval t') and the only element inside pgm will be <oper>. In this call, stackmc pops the top of stk, calculates (<oper> stk.top) and pushes it back to the stk. This value is returned in the next function call, as pgm is shortened by 1. Therefore the final answer returned by (stakcmc stk (compile t)) is equal to <oper> (eval t').

Therefore our inductive step is true for this case.

Case 2: t is of the form <oper> of exptree*exptree, where <oper> is a binary operation.
Let t1 and t2 be the left and right children of t respectively.

Note that both t1 and t2 have heights less than or equal to k, as their parent, t has a height k+1 itself. Therefore, by our inductive assumption,

(eval t1) = (stackmc [] (compile t1)) ……. (1)
(eval t2) = (stackmc [] (compile t2)) ……. (2)

Eval t will equate to (eval t1) <oper> (eval t2)

(Compile t) will operate to a conjunction of lists, (compile t1),(compile t2) and [<oper>] in this order. Let l1 be the length of (compile t1), l2 of (compile t2).

Passing the above to stackmc as pgm, with any arbitrary list as stk, we observe that stackmc will continue modifying stk as per pgm and recursively call itself, each time, shortening the length of pgm by 1.
After l1 function calls, the top most element of stk will be (stackmc [] (compile t1)), which is equal to (eval t1) from eqn 1.
Note that stackmc never modifies the elements initially passed to it in stk, and at the end there is only one new element added to the top of the stk. Therefore, over the next l2 function calls, stk gets another element appended to its top, which will be equal to (stackmc [] (compile t2))=(eval t2) …using (2).
During the next function call, pgm only had <oper> in it, so stackmc pops the top 2 elements from stk, which are (eval t1) and (eval t2) and pushes (eval t1) <oper> (eval t2) on stk. This is returned in the next function call.

Therefore our inductive step is true for this case as well.

Therefore, by induction, we have shown that P(h) holds for all exptrees of height h, where h may be any whole number.

Vidit Jain
2017CS10389