Acroline
Date: / /
Page No:

Proof by induction on $ht(t)$

Base Case: i) $ht(t) = 0$

i.e. $t = N(i)$ for some i.

$$\text{stackmc}(\text{compile}(N(i))) = \text{stackmc}[\text{CONST } i]$$

$$= i$$

stackmc returns the value at the head of the stack. Here we can see that the head of the stack is i and the previous contents of the stack have not been altered

$$\boxed{\text{eval}(N(i)) = i}$$

ii) $ht(t) = 1$

i.e. $t = op(N(i_1), N(i_2))$

where op = Plus, Minus, Mult, Div, Rem [Binary operators].

$$\text{stackmc}(\text{compile}(op(N(i_1), N(i_2)))) = \text{stackmc}([\text{CONST } i_1; \text{CONST } i_2; OP])$$

where OP is the corresponding opcode of op i.e PLUS, etc...

We know that stackmc [CONST i_1] = i_1 which is the value at the head of the stack whose length has been increased by 1. By the time stackmc consumes CONST i_1, (since it is tail recursive) i_1 will be pushed into the stack and by the time it consumes CONST i_2, i_2 will be pushed into the stack on top of i_1. The length of the stack has increased by 2 so far.

$$\text{stackmc}([\text{CONST } i_1; \text{CONST } i_2; OP]) = \text{stackmc}([OP])$$

$$= i_1 \; op \; i_2 = \text{eval}(op(N(i_1), N(i_2)))$$

where op is the corresponding operator for OP.

i_1 and i_2 are popped from the top of the stack and we see that the previous contents of the stack aren't tampered with and the result is pushed onto the stack. (Similarly it can be proved for unary operators).

Induction hypothesis:- Let us assume that for all t' : exptree such that $ht(t') \leq k$:

$$\text{stackmc}(\text{compile}(t')) = \text{eval}(t').$$

and when stackmc terminates length(stack) increases by 1 and previous contents of stack are not altered

IS = i) Let us take the case of binary operators.

Let $ht(t) = k+1$ where

$t = op(t1, t2)$ where t1, t2 are exptrees and
op = Plus | Minus | Mult | Div | Rem

$\Rightarrow \max(ht(t1), ht(t2)) = k.$

$$\text{stackmc}(\text{compile}(op(t1,t2))) = \text{stackmc}(\text{compile}(t1) @ \text{compile}(t2) @ [OP]).$$

We know that when stackmc finishes evaluating compile (t1), length of the stack increases by 1 and the result is pushed onto the stack. Similarly for compile (t2).

We also know that the result of stackmc (compile t1) is (eval t1) and stackmc (compile t2) is (eval t2) (from the induction hypothesis)

We also know that compile t1 will be processed first and then compile t2 and then [OP] because stackmc is a tail recursive function.

$ht\ t1,\ ht\ t_2 \leq k.$

Acroline
Date: / /
Page No:

Hence,

$$stackmc(compile(t1) @ compile(t2) @ [OP]) = stackmc([OP])$$
$$= eval\ t1 \; op \; eval\ t2.$$

Where op = +, -, *, /, mod corresponding to the opcode.

$$eval(Op(t1, t2)) = eval\ t1 \; op \; eval\ t2 \quad \text{by definition.}$$

ii) Similarly for unary operators,

~~stackmc (op(t1))~~

$$stackmc(compile(op(t1))) = stackmc(compile(t1) @ [OP])$$
$$= stackmc([OP])$$
$$= op(eval\ t1)$$

Once stackmc consumes compile t1, the result is pushed onto the stack and since $ht(t1) \leq k$, ~~compi~~ stackmc (compile t1) = eval t1.

Here op = abs, neg. → The corresponding operator for the unary OPCODES.

$$eval(op(t1)) = op(eval\ t1) \quad \text{by definition.}$$

Hence proved.