Mayank Singh Kushwaha
2017CS50413.

Page No.
Date:

To prove :- stackmc (S, compile(t)) = mk_big(eval(t))
where i) t is of type exptree , ii) stackmc (a,b) gives the head of the stac list a after it stops computing
iii) eval (t) gives the final ans of exptree t after which is an int.
iv) mk_big (t) – converts int to bigint

Taking induction on the height of the exptree(t)

Base case :- ht(t) = 0 height of exptree
if h(t) = 0 then t = N(a).
→ compile t gives us a represented as bigint let us give this name a'. a' is a opcode list
→ ~~st~~ Now,

stackmc ([ ]) . (a') = mk_big a

this is by the defination of stackmc.

i.e stackmc [ ] (compile N(a))

= stackmc { [ ], [CONST (mk_big a)]

= mk_big a

Now, mk_big (eval(N(a)))

= mk_big a

hence, mk_big (eval (N(a))) = stackmc [ ] (compile(N(a)))
true for base case.

Scanned by CamScanner

Induction Hypothesis :- for ∀ exptree t of ht less than h our assumption is valid.

i.e ~~h~~ true

$$\text{stackmc}([\,], \text{compile}(t)) = \text{mk\_big}(\text{eval } t)$$

for ∀ t such that $ht(t) \leq h$.

Induction Step : let $ht(t) = h$.

Note :- Plus, Minus, Mult, Div, Rem and corresponding operations in int and bigint are binary opertions while ~~as~~ Nega, Abs and corresponding opertations are unary operations.

case 1 :- let the operation be binary type.

Now in binary operation there are 2 types one which follow commutative property and one which does not.

If we show our property for not commutative operators then it is valid for commutative operators and hence will be true for all operators.

~~so~~ WLOG we ~~so~~ choose subtraction operators (sub)

let $t = \text{Minus}(t_1, t_2)$

s.t. $\max(ht(t_1), ht(t_2)) = h-1$

then $ht(t) = h-1+1 = h$.

By induction hypothesis

~~&~~ stackmc [ ] compile(t_1) = mk_big (eval (t_1)) = r_1

& stackmc [ ] compile(t_2) = mk_big (eval (t_2)) = r_2

Page No.
Date:

LHS :- stackmc [] compile (t) = stackmc [] compile (Minus t_1 t_2) [MINUS]
= stackmc [] (compile t_1 @ compile t_2 @ Minus) [MINUS]
(postorder of opcode list)
= stackmc [r_1] (compile t_2 @ Minus [MINUS])
= stackmc [r_2 ; r_1] [MINUS]
= stackmc [sub r_2 r_1] []
= sub r_1 r_2 = $r_1 - r_2$

RHS :- mk_big (eval t) = mk_big (Minus (t_1, t_2))
= mk_big ((eval t_1) − (eval t_2)) = mk_big ($r_1' - r_2'$)
= ~~tosl~~ $r_1 - r_2$ = LHS. where r_1' = eval t_1
r_2' = eval t_2
hence RHS = LHS.

case 2 :- Let the operation be Abs WLOG.
t = Abs (q t_1) s.t ht(t_1) = h − 1
ht(t) = 1 + ht(t_1) = h.

By induction hypothesis
stackmc [] (compile q) = mk_big (eval q) = r_1
eval q = r_1'

LHS = stackmc [] (compile (Abs (q)))
= stackmc [] (q(compile q) @ [ABS])
= stackmc [r_1] [ABS]
= stackmc [abs r_1] []
= abs r_1 = l (say).

Scanned by CamScanner

Page No.
Date:

RHS : $mk\_big\ (eval\ t) = mk\_big\ (\ eval\ (Abs\ q))$
$= mk\_big(\ abs\_int(\epsilon, v_i'))$
$= \bot\ = LHS.$

hence proved..

Now we have taken an assumption that stackmc [ ] S opcodelist S is [ ] but this can be generalised because we are giving only the head as the output. ~~of the stack~~

Hence we concluded the proof that

stack s (compile (t)) = mk_big (eval(t))