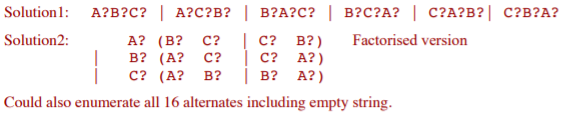
aa1.

a.



b..



C.

Disclaimer: might be very wrong, please correct it or add new solution if necessary

Stat -> Test | TestLess | TestLess moreStat | Test more Stat

Test -> test ‘(’ Expr ‘,’ Expr ‘)’ Stat

TestLess -> Test less Stat

===

What about this?

Stat -> … | Test

Test -> test ‘(‘ Expr ‘,’ Expr ‘)’ Stat TestOptions

TestOptions -> less Stat | less Stat more Stat | more Stat | more Stat less Stat | ε

Third solution: Honestly the above are overly complicated:

S -> … | TestStat

TestStat -> test ‘(‘ E ‘,’ E ‘)’ S X

X -> less S Y | Epsilon

Y -> more S | Epsilon

Can't you just have:

T -> 'test' '(' E ',' E ')' S ['less' S] ['more' S]

Example that grammar is ambiguous

Test (E1, E2) test (E3, E4) S1 less S2 more S3

Could be seen as:

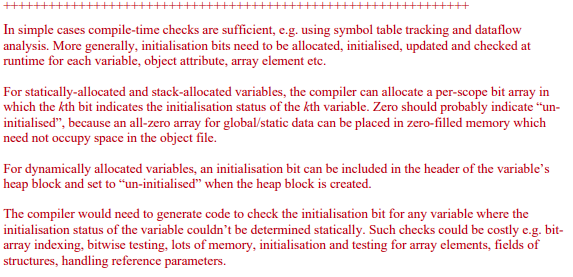
Test (E1, E2) (test (E3, E4) S1) less S2 more S3

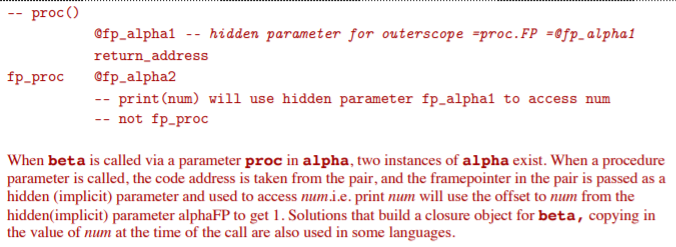
Test (E1, E2) (test (E3, E4) S1 less S2) more S3

Test (E1, E2) (test (E3, E4) S1 less S2 more S3)

D. from the last tutorial

E.





2a. Lines 2 & 10

b. T2

/

T1 - T3 - T0 -> 2 colours => 2 registers

(should be 4 – Himanshu is correct)

\

T4

HERE WE GO AGAIN WITH ANOTHER TRY

T2 - T4 - T0 - T3

\ /

T1

C. T2, because it’s least often accessed or updated in the code

|  |
| --- |
| data Instr = Define String  | Mov Operand Operand  | Add Operand Operand  | Mul Operand Operand  data Operand = Reg Register  | Ind Register  | Abs String  | ImmNum Int  | ImmName String  data Register = D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7  data Stat = Assign Lhs Exp  data Lhs = Var String | Array String Exp  data Exp = Ref Lhs | Plus Exp Exp | Num Int    transExp:: Exp -> [Register] -> [Instr]  transExp (Plus e1 e2) r@(dst:next:rest)  = (transExp e1 r) ++ (transExp e2 (next:rest)) ++  [Add (Reg next) (Reg dst)]  transExp (Num a) (dst:rest)  = [Mov (ImmNum a) (Reg dst)]  transExp (Ref lhs) r  = transLhs lhs r  transLhs:: Lhs -> [Register] -> [Instr]  transLhs (Var str) (dst:rest)  = [Mov (Abs str) (Reg dst)]  transLhs (Array str exp) r@(dst:rest)  = (transExp exp r) ++ [Mul (ImmNum 4) (Reg dst)] ++  [Add (Abs str) (Reg dst)] ++ [Mov (Ind dst) (Reg dst)]  transStat :: Stat -> [Register] -> [Instr]  transStat (Assign l e) r@(dst:next:rest)  = (transLhs l r) ++ (transExp e (next:rest))  ++ [Mov (Reg next) (Reg dst)] |

I did something like this, if you find there is a problem, please [@Liu, Taowen](mailto:tl2020@ic.ac.uk)

I know that there are several registers that I can reuse, but I am just being lazy

transExp :: Exp -> [Register] -> [Instr]

transExp (Ref (Var str)) (val:\_)

= [Mov (ImmName str) (Reg val)]

transExp (Ref array) (val:addr:r)

= getArrAddr array (addr:val:r) ++

[Mov (Ind addr) (Reg val)]

transExp (Plus e1 e2) (fst:snd:rs)

= transExp e1 (fst:snd:rs) ++

transExp e2 (snd:rs) ++

[Add (Reg snd) (Reg fst)]

transExp (Num imm) (val:\_)

= [Mov (ImmNum imm) (Reg val)]

transStat :: Stat -> [Register] -> [Instr]

transStat (Assign (Var str) exp) (src:rs) =

transExp exp (src:rs) ++

[Mov (Reg src) (Abs str)]

transStat (Assign array exp) (src:addr:rs) =

transExp exp (src:rs) ++

getArrAddr array (addr:rs) ++

[Mov (Reg src) (Ind addr)]

getArrAddr (Array str exp) (dst:offset:r)

= transExp exp (offset:r) ++

[Mul (ImmNum 4) (Reg offset),

Mov (Abs str) (Reg dst),

Add (Reg offset) (Reg dst)]

(f) The range of the for loop is typically fixed. We could exploit the array subscript as an induction variable and perform bounds checks on every iteration.