Square Root by Long Division

- ## Functions

Note: - Lists with variable names ending in "__" are to be read from left to right; hence last digit is the last element of the list and the rest from right to left; hence last digit is the first element of the list
- num(var) represents the numerical value of var
- lis(var) represents the list equivalent of var
- Append means adding to the front of the list
- in algorithms, branching is a part of the else block if not mentioned explicitly
- "rest of the list" denotes the tail of the list

- ### Athematic Functions
  - #### Addition

1. digit = fn: int list -> int

Gives the first element for a non-empty list and 0 for an empty list.

$$\text{digit(l)=} \begin{cases} \text{first element of list} & \text{: if list is not empty} \\ 0 & \text{: if lis is empty} \end{cases}$$

Proof of Correctness : Correct by defination

2. headlesslist = fn : 'a list -> 'a list

Gives the tail for a non empty list and an empty list for an empty list

$$\text{headlesslist(l)=} \begin{cases} \text{Empty list} & \text{: if lis is empty} \\ \text{list without first element} & \text{: if list is not empty} \end{cases}$$

Proof of Correctness : Correct by defination

3. add_digits = fn : int * int * int -> int

Gives the unit digit of the sum of x,y and c

add_digits(x,y,c)= (x + y + c) % 10

Proof of Correctness: Correct by defination

4. carry_in_addition = fn : int * int * int -> int

Gives the tens digit of the sum of x,y and c

carry_in_subtraction(x,y,c)= (x + y + c) / 10

Proof of Correctness: Correct by defination

5. add = fn : int list * int list * int -> int list

Recursively adds two numbers represented as lists by adding heads of the list and passing rest of the elements of the lists along with a carry to itself until both lists are empty.

$$\text{add(l1,l2,c)=} \begin{cases} \text{Empty list : if both lists are empty and carry is zero} \\ \text{Singleton list with carry as the element : if both lists are empty and carry is not zero} \\ \text{Append the unit digit of sum of first digits of both lists : otherwise} \\ \text{to the sum of rest of the lists along with carry} \end{cases}$$

Proof of Correctness: Proof by Induction assuming helper functions are correct

Induction varriable, n= length of the longer list

Base Case : n=0, then both numbers are 0, thus if there is some carry the result is the list equivalent of carry and if there is no carry the result is empty list which is equivalent to 0. Hence, add() is correct for n=0.

Induction Hypothesis: If add() is correct for n<n' then add() is correct for n=n'

Let l1'and l2' denote the rest of the lists for which n=n'-1
Hence num(l1')=num(l1) div 10 and num(l2')=num(l2) div 10
c'=tens digit of sum of last digit with carry added
Unit digit of sum of last digits u = num(l1) mod 10 + num(l2) mod 10 - 10*c' +c
Hence num(add(l1',l2',c))= num(l1') + num(l2')= num(l1) div10 + num(l2) div 10 + c' as add is correct for n<n'
num(add(l1,l2,c))= num(Append u to add(l1',l2',c'))=10 * num(add(l1',l2',c')) +u
= 10 * (num(l1) div 10) + 10 * (num(l1) div 10) +num(l1) mod 10 + num(l1) mod 10 +c
= num(l1) + num(l2) + c
Hence, add() is correct for n=n'
Hence add() is correct for all n

Subtraction

6. subtract_digits = fn : int * int * int -> int

Gives the unit digit of the difference of x+c and y

subtract_digits(x,y,c)= (x - y + c) % 10

Proof of Correctness: Correct by defination

7. carry_in_subtraction = fn : int * int * int -> int

Gives the tens digit of the difference of x+c and y

carry_in_subtraction(x,y,c)= (x - y + c) / 10

Proof of Correctness: Correct by defination

8. dirty_subtract = fn : int list * int list * int -> int list

Recursively subtractss two numbers represented as lists by adding heads of the list and passing rest of the elements of the lists along with a carry to itself until both lists are empty.

dirty_subtract(l1,l2,c)= { Empty list : if both lists are empty
Append the unit digit of difference of first digits of both lists to the sum of rest of the lists along with carry : otherwise

Proof of Correctness: Proof by Induction assuming helper functions are correct

Induction varriable, n= length of the longer list

Base Case : n=0, then both numbers are 0, the result is empty list which is equivalent to 0. Hence, dirty_subtract() is correct for n=0.

Induction Hypothesis: If dirty_subtract() is correct for n<n' then dirty_subtract() is correct for n=n'

Let l1'and l2' denote the rest of the lists for which n=n'-1
Hence num(l1')=num(l1) div 10 and num(l2')=num(l2) div 10
c'=tens digit of difference of last digit with carry added(negative if carry is borrowed)
Unit digit of difference of last digits u with carry added = num(l1) mod 10 - num(l2) mod 10 - 10*c' + c
Hence num(dirty_subtract(l1',l2',c))= num(l1') + num(l2')= num(l1) div10 - num(l2) div 10 + c'
as dirty_subtract() is correct for n<n'
num(dirty_subtract(l1,l2,c))= num(Append u to add(l1',l2',c'))=10 * num(subtract(l1',l2',c')) +u
= 10 * (num(l1) div 10) - 10 * (num(l1) div 10) +num(l1) mod 10 - num(l1) mod 10 +c
= num(l1) - num(l2) + c
Hence,dirty_subtract() is correct for n=n'
Hence dirty_subtract() is correct for all n

9. clean = fn : int list -> int list

Removes unnecessary zeros from the end of the list recursively until the last element of the list is is non-zero

$$\text{clean(l)} = \begin{cases} \text{Empty list : if list is empty} \\ \begin{cases} \text{Empty list : if the rest of the list is empty after cleaning and first element is 0} \\ \text{Apeend first element to clean form of rest of the list : otherwise} \end{cases} \end{cases}$$

Proof of Correctness: Proof by Induction assuming helper functions are correct

Induction varriable, n= length of the list

Base Case : n=0,the result is empty list, thus there are no extra zeros. Hence clean() is correct for n=0

Induction Hypothesis: If clean() is correct for n<n' then clean() is correct for n=n'

Let l' denote the rest of the lists for which n=n'-1
Hence clean(l') has no extra zeros as clean() is correct for n<n'
Proof by cases:
If clean(l') is empty, the first element of l should be the first non zero digit of num(l)
Hence if first element of l is zero it should be removed and the result which is an empty list has no extra zeros and if first element is not zero ist is the first non zero digit and result is a singleton list with only the first element of l and thus has no extra zeros.
If clean(l') is not empty then there is already a non zero leading digit thus apeend of first element of l to clean(l') has no extra zeros.
Hence clean(l') has no extra zeros.
Thus clean() is correct for n=n'
Hence clean() is correct for all n.

10. subtract = fn : int list * int list * 'a -> int list

subtracts two numbers represented as lists and also removes unnecessary zeros

subtract(l1,l2,c)= clean(dirty_subtract(l1,l2,c))

Proof of Correctness: Correct due to correctness of clean() and dirty_subtract()

- Comparison

11. compare_adv = fn : int list * int list -> int

compares two numbers represented by lists of equal length recursively by comparing the later digits first and if they are equal then comparing the first elements.

(Key => 0: > , 1: < , 2: =)

$$\text{compare\_adv(l)=} \begin{cases} \text{2 : if the list are empty} \\ \begin{cases} \text{Numerical representation of comparasion between first elements} \\ \text{of both lists : If l1 and l2 are equal except at unit digits} \\ \text{Comparasion of the rest of the lists : if the rest of the lists are not equal} \end{cases} \end{cases}$$

Proof of Correctness: Proof by Induction

Induction varriable, n= length of the lists

Base case : n=0, the lists are empty and are therefore equal

Induction Hypothesis: If compare_adv() is correct for n<n’ then compare_adv() is correct for n=n’

Let l1’,l2’ denote the rest of the lists for which n=n’-1
Hence num(l1’)=num(l1) div 10 and num(l2’)=num(l2) div 10
Hence compare_adv(l1’,l2’) is correct as compare_adv() is correct for n<n’
Proof by cases:
If compare_adv(l1’,l2’) equals 0 , num(l1) div 10 > num(l2) div 10 hence num(l1)>num(l2)
Thus compare_adv(l1,l2)= compare_adv(l1’,l2’)=0
If compare_adv(l1’,l2’) equals 1 , num(l1) div 10 < num(l2) div 10 hence num(l1)<num(l2)
Thus compare_adv(l1,l2)= compare_adv(l1’,l2’)=1
If compare_adv(l1’,l2’) equals 2 , l1 and l2 are equal at all places other than unit digit
Thus compare_adv(l1,l2)= numerical representation of comparasion first elements of l1 and l2
Hence compare_adv(l1,l2)=numerical representation of relation between l1 and l2
Thus compare_adv() is correct for n=n’

12. compare = fn : int list * int list -> bool

Compares two numbers represented as lists and gives the value of l1>l2

$$\text{compare(l1,l2)=} \begin{cases} \text{True : if l1 is longer than l2} \\ \text{False : if l2 is longer than l1} \\ \text{compare\_adv(l1,l2) : if length of l1 and l2 are equal} \end{cases}$$

Proof of Correctness: Proof by Cases

If length(l1) > length(l2) then num(l1)>num(l2) hence result is true
If length(l2) > length(l1) then num(l2)>num(l1) hence result is false
If length(l1) = length(l2) then the result is compare_adv(l1,l2) as compare_adv() is correct

- Multiplication

13. multiply_digits = fn : int * int * int -> int

    Gives the unit digit of the product of x and y added to c

    multiply_digits(x,y,c)= (x * y + c) % 10

    Proof of Correctness: Correct by defination

14. carry_in_multiplication = fn : int * int * int -> int

    Gives the tens digit of the product of x and y added to c

    carry_in_multiplication(x,y,c)= (x * y + c) / 10

    Proof of Correctness: Correct by defination

15. multiply = fn : int list * int * int -> int list

    multiplies a number represented as a list and a single digit number recursively by multiplying the first element and passing rest of the digits to itself along with a carry

multiply(l,n,c)= {

- Empty list : if both list is empty and carry is zero
- Singleton list with carry as the element : if both list is empty and carry is not zero
- Append the unit digit of product of lfirst digits of the list : otherwise
  to the product of rest of the list and n along with carry added

Proof of Correctness: Proof by Induction assuming helper functions are correct

Induction varriable, n= length of the list

Base case : n=0, the list is empty and therefore product with n is zero and the result is an empty list

Induction Hypothesis: If multiply() is correct for n<n’ then multiply() is correct for n=n’

Let l’ denote the rest of the lists for which n=n’-1
Hence num(l1’)=num(l1) div 10 and num(l2’)=num(l2) div 10
Hence multiply(l’,n,c’) = num(l’)*n + c’ = (num(l) div 10) *n +c’ as multiply() is correct for n<n’
c’=tens digit of product of last digit with carry added
Unit digit of product of first element and n with carry added= (num(l) mod 10)*n - 10 *c’ +c
num(multiplyt(l,n,c))= num(Append u to multiply(l’,n,c’))=10 * num(multiply(l’,n,c’)) +u
    = 10 * (num(l) div 10) *n +(num(l) mod 10)*n +c
    = num(l)*n + c
Hence,multiply() is correct for n=n’
Thus multiply() is correct for all n

- ## Conversion Functions

16. convert_to_digit = fn : char -> int

converts a digit in character form to its numerical value

convert_to_digit(x)= ord(x) - 48

Proof of Correctness: Correct by defination

17. convert_to_character = fn : int -> char

converts a digit to character

convert_to_character(n)= chr(n+48)

Proof of Correctness: Correct by defination

18. change = fn : char list -> int list

changes a list of characters to a list of their corresponding digits

change(l)= {
- Empty list : if the list isempty
- Append the digit form of first element of the list to the list of list of digits equivalent to the rest of the elements : otherwise

Proof of Correctness: Proof by Induction assuming helper functions are correct

Induction varriable, n= length of the list

Base case : n=0, the list is empty and therefore the result is an empty list as there are no elements to change

Induction Hypothesis: If change() is correct for n<n' then change() is correct for n=n'

Let l' denote the rest of the lists for which n=n'-1
Hence change(l') gives a list of rest of the elements of l converted to digits as change() is correct for n<n'
Helce appending digit form of first element of l to change(l') gives a list of elements of l converted to l' as convert_to_digit() is correct
Hence change() is correct for n=n'
Hence change() is correct for all n

19. convert = fn : string -> int list

Converts a string to a list of characters

convert(str)= change(String.explode(str))

Proof of Correctness: Correct due to correctness of change()

20. change_back_and_reverse = fn : int list * char list -> char list

Changes a list of integers to a reversed list of characters and appendss it to the second list

$$\text{change\_back\_and\_reverse}(x,y)= \begin{cases} y : \text{if x is empty} \\ \\ \text{Append the character form of first : otherwise} \\ \text{element of x to y and remove it from x} \\ \text{And pass the new lists to itself as parameters} \end{cases}$$

Proof of Correctness: Proof by Induction assuming helper functions are correct

Induction varriable, n= length of x
Base case : n=0, x is empty and therefore the result is y as there are no elements to change back

Induction Hypothesis: If change_back_and_reverse() is correct for n<n' then change_back_and_reverse() is correct for n=n'

Let x' denote the rest of x for which n=n'-1
New second list y'=Append changed first element to y
Hence change_back_and_reverse(x',y') gives a reversed and converted x' appended to y' as change_back_and_reverse() is correct for n<n'
Thus change_back_and_reverse(x',y')= reverse(x')::y'
= reverse(x') :: first element of x :: y = reverse(x) :: y
Hence change_back_and_reverse() is correct for n=n'
Hence change_back_and_reverse() is correct for all n

21. convert_back = fn : int list -> string

Converts a list of digits into its corresponding string representation

$$\text{convert\_back(l)=} \begin{cases} \text{0 : if list is empty} \\ \text{implode(change\_back\_and\_reverse(l,[])) : otherwise} \end{cases}$$

Proof of Correctness: Correct due to correctness of change_back_and_reverse

- ## Find Digit Functions

22. check_equation = fn : int * int list * int list -> bool

Gives the value of (digit*digit + bulk*digit*10) <= remainder

check_equation(digit,remainder,bulk)=not(compare(multiply(digit::bulk,digit,0),remainder))

Proof of Correctness: Correct due to correctness of compare() and multiply()

23. loop = fn : int list * int list * int -> int

Returns the largest single digit number less than given digit for which check_equation() gives true

$$\text{loop(bulk,remainder,digit)=} \begin{cases} \text{digit : if check\_equation(digit,remainder,bulk) is true} \\ \text{loop(bulk,remainder,digit-1) : otherwise} \end{cases}$$

Proof of Correctness: Proof by Induction assuming helper functions are correct

Induction variable: digit

Induction Hypothesis : If loop() is correct for digit<digit' then loop() is correct for digit=digit'

Base case: digit= dig such that check_equation() gives true

Let digit=digit'
If check_equation() gives true the result is digit'
else result equals loop() for digit'-1
As loop() is correct for digit'-1 , result<=digit'-1
Hence result<=digit'
Thus loop() is correct for digit=digit'
Therefore, loop() is correct for all digit

24. find_digit = fn : int list * int list -> int

Finds the digit to be merged into bulk as the new unit digit

find_digit(bulk,remainder)= loop(bulk,remainder,9)

Proof of Correctness: Correct due to correctness of loop()

- Long Division Functions

25. update_varriables = fn : int list * int list * int list * int list -> string * string

This function recursively performs the long division update_varriables(b,r,a,n_)=

- Answer as a pair of strings after adding the new digit to answer and subtracting dig*(num(bulk)*10+dig) from the remainder : if number__ is empty
- Add new digit to answer,change bulk to lis(10*num(bulk)+2*dig remainder to lis((num(remainder)-dig*(num(bulk)*10+dig))*100 + next two digits) and number__ to lis(num(number__) div 100)

  : otherwise

Proof of Correctness: Proved collectively with the calculate_squareroot() function

26. calculate_squareroot = fn : int list -> string * string

Calculates the nearest integer squareroot and the remainder of a number represented as a

calculate_squareroot(l)=

- ("0","0") : if the list is empty
- Call update_varriables() starting with only the first digit as remainder and rest of number__ : if length of the list is odd
- Call update_varriables starting with first two digits of number__ as remainder and remove these from number__ : if length of the list is even

Proof of Correctness: Proof that calculate_squareroot() and update_varriables() collectively are analogous to the long division method

Hence, assuming the correctness of longdivision method, calculate_squareroot() and update_varriables() are correct

The long division groups the digits in pairs of two and starts with single or two digits depending on weather there are odd or even number of digits. This is done by the calculate_squareroot() function which then calls the update_varriables() function with appropriate initialisation. Hence update_varriables() alway has even length for the parameter: number__.

The cases for length(l) equal to 0 and 1 are handeled by if elseif block in calculate_squareroot()

Correctness of update_varriables() : Proof by logical deduction

Induction Varriable: n_=len(number__)/2

Base Case: n_=0, num(answer)= num(answer)*10 +maximum value of digit such that (num(bulk)+digit)*digit is less than remainder as initial values of answer and bulk are zero
num(answer) = maximum value of digit such that digit*digit< less than remainder
Thus the function is correct for n_=0

Lemma: for a number with n digits with di being the ith digit(from left),

number^2= $\sum$ di*(100^(i-1)) + $\sum$ di*dj*(10^(i+j-2))

Induction Hypothesis: If assumption is correct for n<n', it is correct for n=n'

Assumption: if di represent the k digits added after n=n_1 step from last then,

remainder'= $\sum$ di*bulk'*(10^(n+I-1)) + $\sum$ di*(100^(i-1)) + $\sum$ di*dj*(10^(i+j-2))
where remainder'and bulk' are remainder and bulk provided to the next step

If for the current step, the digit chosen is d(k+1), then remainder=remainder + remainder'
As bulk'=10*bulk+2*digit
remainder= $\sum$ di*bulk*(10^(n+1+I-1)) + $\sum$ (di^2)*(100^(i-1)) + $\sum$ di*dj*(10^(i+j-2)) + d(k+1)^2*100^n + 2 $\sum$ d(k+1)*di*(10^(n+i-1)
= $\sum$ di*bulk*(10^(n+1+I-1)) + $\sum$ (di^2)*(100^(i-1)) + $\sum$ di*dj*(10^(i+j-2)) with i going from 1 to k+1

Which is equal to the remainder passed from previous step i.e. n=n_+1 th step from last
Hence if aaumption is true for all valuse of n and therefore for all values of n_

Thus, if (a,r) is the answer from update_varriables() then ,
number= $\sum$di*bulk*(10^(n+1+I-1)) + $\sum$(di^2)*(100^(i-1)) + $\sum$di*dj*(10^(i+j-2)) + r
Where di are the digits of a from left
As bulk=0 initially
number= $\sum$(di^2)*(100^(i-1)) + $\sum$di*dj*(10^(i+j-2)) + r = a^2 + r (by lemma)
Hence, the algorithm is correct.

- ## Main Function

27. isqrld = fn : string -> string * string

calculates the nearest integer squareroot and the remainder of a number represented as a string

isqrtld(str)=calculate_squareroot(convert(str))

Proof of Correctness: Correct due to correctness of convert() and calculate_squareroot()

Submitted by: Anshik Sahu
2021CS10577