CS3723 Pgm 4 LISP Derivative (50 pts)

© 2018 Larry W. Clark, this document may not be copied to any other website.

Since LISP processes lists, provides recursion, and can dynamically change between data and code, it is well suited for differentiation. In this assignment, you will provide functions to create well-formed formulas (WFFs) and symbolically take their derivatives.

**Any use of code from another web site will result in a 0 on this assignment, may result in an F for this course, and may cause you to be expelled from UTSA.**

WFF (20 points)

* Create a function (WFF *inExpr*) which converts an infix-like parenthesized expression to a WFF. (See the examples below.)
* You will be provided with expressions which must be converted to WFFs.
* A polynomial term in the form of (3 X 2) needs to be changed to (P 3 X 2). The term (2 X) needs to be changed to (P 2 X 1). P means polynomial term.
* We will assume that we are not allowed to have (3 g(x) 2). This was done to reduce work for you.
* An infix binary operator (g(x) *binaryOp* h(x)) must be converted to  
   (*binaryOp* (WFF g(x) ) (WFF (h(x)) )

Example: (WFF '((2 X 3) + (5 x 2))) returns (+ (P 2 X 3) (P 5 X 2))

* A unary operator (*unaryOp* g(x)) must be converted to   
  (*unaryOp* (WFF g(x)))
* A constant is not changed. (e.g., the constant 5 is already in its proper form)
* An X by itself is not changed. (e.g., X by itself is already in its proper form). This makes it easier to handle (COS X) being (COS X) instead of (COS (P 1 X 1)). In all other contexts, I will give you (1 X) or (1 X 1) which would be (P 1 X 1).
* Hint: (**this isn't everything to create a WFF**, but it is a huge hint):
  + if the expr is an atom, return it. (e.g., 5, X)
  + examine the CAR
    - if it is a number, it is one of these (examples shown):
      * (3 X 2)
      * (3 X)
      * (3 + g(x)) ; of course, it could have been other binary ops
    - If it is an atom, it is a unary operator
      * If it is a '-, use 'U- as the unary minus operator.
      * Since it is a unary operator, its *cadr* needs to be converted to a WFF.
    - Otherwise, it is a binary operator and both of its binary operands need to be converted to WFFs
* Note that *inExpr* minus is both unary and binary. When converted to a WFF, the unary minus should be shown as 'U-.
* I strongly advise you to create comments within this function to help let you know what it is doing. Example:  
   ;;; check for unary minus  
   ( your lisp code )  
   ;;; otherwise, it is binary  
   ( T your lisp code )

Examples:

|  |  |  |
| --- | --- | --- |
| Seq | Original Expressions | WFFs |
| 1 | ( 4 X 3 ) | (P 4 X 3) |
| 2 | ((2 X 3) + (5 x 2) ) | (+ (P 2 X 3) (P 5 X 2)) |
| 3 | ( ( 3 X 3) + 5) | (+ (P 3 X 3) 5) |
| 4 | ( (4 X 3) + (6 X) ) | (+ (P 4 X 3) (P 6 X 1)) |
| 5 | ( (5 X 4) + ( (3 X 2) + (5 X) ) ) | (+ (P 5 X 4) (+ (P 3 X 2) (P 5 X 1))) |
| 6 | (SIN X) | (SIN X) |
| 7 | ( SIN ( 7 X 2) ) | (SIN (P7 X 2)) |
| 8 | ( (8 X 3) + (SIN (2 X)) ) | (+ (P 8 X 3) (SIN (P 2 X 1))) |
| 9 | ( (9 X 4) - (2 X 3)) | (- (P 9 X 4) (P 2 X 3)) |
| 10 | ( (10 X 3) \* (5 X 2)) | (\* (P 10 X 3) (P 5 X 2)) |
| 11 | ( (11 X 3) / (2 X 1) ) | (/ (P 11 X 3) (P 2 X 1)) |
| 12 | ( (12 X 4) \* ( (1 X 5) + (2 X 4)) ) | (\* (P 12 X 4) (+ (P 1 X 5) (P 2 X 4))) |
| 13 | (SIN ( ( 13 X 2) / (2 X 3)) ) | (SIN (/ (P 13 X 2) (P 2 X 3)) ) |
| 14 | (COS ( 14 X)) | (COS (P 14 X 1)) |
| 15 | ( (15 X 1) - (15 X)) | (- (P 15 X 1) (P 15 X 1)) |
| 16 | (((16 X) + 5) \* (4 X)) | (\* (+ (P 16 X 1) 5) (P 4 X 1)) |
| 17 | ( - (17 X 2)) | (U- (P 17 X 2)) |
| 18 | ( - (COS ( 18 X)) ) | (U- (COS (P 18 X 1))) |

> (WFF '( (3 X 2) + (6 X) ))

(+ (P 3 X 2) (P 6 X 1))

**DIFFERENTIATION (30 pts)**

Taking derivatives of WFFs can involve invoking the appropriate derivative function for a WFF.

Some Derivatives

|  |  |  |
| --- | --- | --- |
| Seq | WFFs | Derivative |
| 1 | (+ (P 4 X 2) (P 6 X 1)) | (+ (P 8 X 1) 6) |
| 2 | (+ (P 2 X 4) (+ (P 3 X 2) (P 5 X 1))) | (+ (P 8 X 3) (+ (P 6 X 1) 5)) |
| 3 | (+ (P 2 X 3) 5) | (+ (P 6 X 1) 0)  … (P 6 X 1) - extra credit for simplifications |
| 4 | (SIN X) | (COS X) |
| 5 | (COS X) | (U- (SIN X)) |
| 6 | (P 6 X 2) | (P 12 X 1) |
| 7 | (P 7 X 1) | 7 |

I have provided functions HADD and HGET which respectively add and get entries from a LISP hash table. We will use it to know the derivative function for each of our WFF functions. We can get the derivative function for a WFF function *fn* by using (HGET *fn*).

We can define a property for a function like P that specifies the derivative function (e.g., DER\_P).

(HADD 'P 'DER\_P)

(HADD '+ 'DER\_+)

You must create derivative functions for each of the WFF functions:

|  |  |
| --- | --- |
| WFF Function | Derivative Function |
| P | DER\_P |
| + | DER\_+ |
| - | DER\_- |
| U- | DER\_U- |
| / | DER\_/ |
| \* | DER\_\* |
| COS | DER\_COS |
| SIN | DER\_SIN |

You must also create a function DERIV which takes the derivative for any WFF.

* A derivative of a constant is 0.
* You have to decide whether you want every derivative function to receive a list of its arguments as a single parameter or receive the arguments separately into different parameters. I recommend passing a list of its arguments.
* The derivative of a WFF function is determined by calling the appropriate derivative function. DERIV will need to use HGET to get the derivative function name and then use EVAL to evaluate it on an appropriate expression. Example:   
   (eval `( , (HGET (car expr)) ' , (CDR EXPR))) ;;; generates (eval *derivFn* ' ( arg1 ))  
   ;;; or (eval *dervFn '* ( arg1 arg2 ))
* Since differentiation is recursively defined, DERIV calls appropriate DER\_xxx functions which might then call DERIV

If always passing a list of parameters, DERIV passes the appropriate function a list of its parameters. Examples of what DERIV passes to the derivative function

|  |  |  |  |
| --- | --- | --- | --- |
| WFF | Derivative Function invoked by DERIV | Evaluated Expression in DERIV | Parameter Received by Derivative Function |
| (SIN X) | DER\_SIN | (DER\_SIN '(X)) | (X) |
| (SIN (P 3 X 2)) | DER\_SIN | (DER\_SIN '((P 3 X 2))) | ((P 3 X 2)) |
| (+ (P 3 X 2) 5) | DER\_+ | (DER\_+ '((P 3 X 2) 5)) | ((P 3 X 2) 5) |

**DERIVATIVE FORMULAS (from Calculus I)**

|  |  |  |
| --- | --- | --- |
| Rule Name | Expression | Derivative Formula |
| Power and Constant Rule | g(x) = cXn | g'(x) = cnXn-1 |
| Sum Rule | g(x) + h(x) | g'(x) + h'(x) |
| Difference Rule | g(x) – h(x) | g'(x) – h'(x) |
| Unary Minus Rule | - g(x) | - g'(x) |
| Product Rule | g(x) \* h(x) | g(x)h'(x) + g'(x)h(x) |
| Chain Rule | f(P(x)) | f'(P(x))\*g'(x) |
| Quotient Rule | g(x) / h(x) | g'(x)h(x)-g(x)h'(x)  (h(x))2 |
| COS | COS (x) | -SIN(x) |
| SIN | SIN(x) | COS(x) |
| Chain Rule with SIN | SIN(P(x)) | COS(P(x)) \* g'(x) |

Here is example code for taking the derivative for U- assuming the parameter is a list of WFFs:

;;; DER\_U-

;;; Parameters:

;;; expr – list around a single wff expression

;;; Purpose:

;;; Returns the derivative of a unary minus expression.

;;; Note:

;;; The derivative of (U- wff) is (U- (deriv wff))

(defun derU- (expr)

(list 'U- (deriv (car expr))) ;;; (u- (deriv wff))

)

I provided two LISP files:

**p4LispDef.txt** – execute this first (before loading your functions)

* It defines:
  + HADD – function to add a value for a key in the DERIV-HT hash table
  + HGET – given a hash key, it returns the value from the DERIV-HT hash table
  + DERIV-HT – hash table for storing the functions and corresponding derivation functions

**p4LispRun.txt** – execute this after you have loaded your functions

* It uses your WFF function to convert infix-like expressions to WFFs. It then saves those in variables.
* It uses your DERIV function to take the derivatives of those WFFs.

To execute your code:

(load "p4LispDef.txt" :echo T :print T)

(load "p4Lisp.txt" :echo T :print T) ;;; your code

(load "p4LispRun.txt" :echo T :print T) ;;; execute test cases

**EXTRA CREDIT**

I have provided two extra credits where you will receive at least 13 points by doing both. Let's say 10 people do EC#1 (and receive full credit) and 2 people do EC#2 (and receive full credit). Those two people would receive:

EC#1: 5 + 200/10 = 25

EC#2: 5 + 100/2 = 55

Total: 80 pts

To be eligible for any of the extra credit, the assignment **must not be late** and must follow my programming standards.

Do **not** simplify the original WFFs.

**EXTRA CREDIT #1** (points = 5 + 200/n where n is the total number of people (from both classes) with FULL credit on EC #1)

Simplify resulting derivatives. Some examples:

|  |  |
| --- | --- |
| Resulting Derivative Without Simplification | Simplified |
| (+ (P 3 X 2) 0) | (P 3 X 2) |
| (+ 0 (P 3 X 2)) | (P 3 X 2) |
| (- (P 5 X 2) (P 5 X 2)) | 0 |
| (- 2 2) | 0 |
| (+ (P 6 X 3) (P 7 X 3)) | (P 13 X 3) |
| (U- (U- (P 5 X 2))) | (P 5 X 2) |
| (+ (P 3 X 2) (- (P 5 X 2) (P 5 X 2))) | (P 3 X 2) |

You must be able to do all of those (or similar ones) to get "full credit".

**EXTRA CREDIT #2** (points = 5 + 100 / m where m is the total number of people (from both classes) on EC#1 and EC #2)

You must also have done extra credit #1 and received full credit. You must be able to demonstrate that your code can simplify derivatives in these forms:

(U- (\* (U- WFF1) WFF2)) to (\* WFF1 WFF2)

(U- (\* WFF1 (U- WFF2))) to (\* WFF1 WFF2)

(\* (U- WFF1) (U- WFF2)) to (\* WFF1 WFF2)

|  |  |
| --- | --- |
| Resulting Derivative Without Simplification | Simplified |
| (U- (\* (U- (P 3 X 2)) (P 5 X 3))) | (\* (P 3 X 2) (P 5 X 3)) |
| (U- (\* (P 3 X 2) (U- (P 5 X 3)) )) | (\* (P 3 X 2) (P 5 X 3)) |
| (\* (U- (P 3 X 2)) (U- (P 5 X 3))) | (\* (P 3 X 2) (P 5 X 3)) |

For either EC#1 or EC#2 full credit, your code must work for all the other cases. (In other words, your inclusion of code to simplify shouldn't cause your other code to break.)

**What to turn in?**

* Run your code on a fox server to make certain it works.
* Turn in a zip file containing:
  + Your LISP code named p4Lisp.txt which must follow my programming standards
  + A log of your CLISP session.
* A comment in Blackboard to the TA specifying one of the following (failure to specify this will unfortunately make you ineligible for extra credit):
  + NO Extra Credit
  + Only Extra Credit #1
  + Both Extra Credit