CS3723 Pgm 4 LISP Differentiation (50 pts)

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.

WFF

* 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 (G 3 X 2). The term (2 X) needs to be changed to (G 2 X 1).
* We will assume that we are not allowed to have (3 g(x) 2). This was done to reduce work.
* An infix binary operator (g(x) *binaryOp* h(x)) must be converted to  
   (*binaryOp* (WFF g(x) ) (WFF (h(x)) )
* 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 (G 1 X 1)). In all other contexts, I will give you (1 X) or (1 X 1) which would be (G 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 CAR 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 minus is both unary and binary. When converted to a WFF, the unary minus should be shown as 'U-.

Examples:

|  |  |  |
| --- | --- | --- |
| Seq | Original Expressions | WFFs |
| 1 | ( 4 X 3 ) | (G 4 X 3) |
| 2 | ((2 X 3) + (5 x 2) ) | (+ (G 2 X 3) (G 5 X 2)) |
| 3 | ( ( 3 X 3) + 5) | (+ (G 3 X 3) 5) |
| 4 | ( (4 X 3) + (6 X) ) | (+ (G 4 X 3) (G 6 X 1)) |
| 5 | ( (5 X 4) + ( (3 X 2) + (5 X) ) ) | (+ (G 5 X 4) (+ (G 3 X 2) ( G 5 X 1))) |
| 6 | (SIN X) | (SIN X) |
| 7 | ( SIN ( 7 X 2) ) | (SIN (G7 X 2)) |
| 8 | ( (8 X 3) + (SIN (2 X)) ) | (+ (G 8 X 3) (SIN (G 2 X 1))) |
| 9 | ( (9 X 4) - (2 X 3)) | (- (G 9 X 4) (G 2 X 3)) |
| 10 | ( (10 X 3) \* (5 X 2)) | (\* (G 10 X 3) (G 5 X 2)) |
| 11 | ( (11 X 3) / (2 X 1) ) | (/ (G 11 X 3) (G 2 X 1)) |
| 12 | ( (12 X 4) \* ( (1 X 5) + (2 X 4)) ) | (\* (G 12 X 4) (+ (G 1 X 5) (G 2 X 4))) |
| 13 | (SIN ( ( 13 X 2) / (2 X 3)) ) | (SIN (/ (G 13 X 2) (G 2 X 3)) ) |
| 14 | (COS ( 14 X)) | (COS (G 14 X 1)) |
| 15 | ( (15 X 1) - (15 X)) | (- (G 15 X 1) (G 15 X 1)) |
| 16 | (((16 X) + 5) \* (4 X)) | (\* (+ (G 16 X 1) 5) (G 4 X 1)) |
| 17 | ( - (17 X 2) | (U- (G 17 X 2)) |
| 18 | ( - (COS ( 18 X)) ) | (U- (COS (G 18 X 1))) |

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

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

**DIFFERENTIATION**

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

Some Derivatives

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

I have provided functions PUTP and GETP.

We can define a property for a function like G that specifies the derivative function (e.g., DERG).

(PUTP 'G DERIVATIVE 'DERG)

(PUTP '+ DERIVATIVE 'DER+)

We can get the derivative function for a function *fn* by using (GETP *fn* DERIVATIVE).

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

|  |  |
| --- | --- |
| WFF Function | Derivative Function |
| G | DERG |
| + | DER+ |
| - | DER- |
| U- | DERU- |
| / | DER/ |
| \* | DER\* |
| COS | DERCOS |
| SIN | DERSIN |

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

* A derivative of a constant is 0.
* The derivative of a WFF function is determined by calling the appropriate derivative function. DER will need to use GETP to get the derivative function name and then use EVAL to evaluate it on an appropriate expression. You will probably need to quote the argument to the derivative function. This can be done by stating:  
   (LIST ' ' (CDR EXPR)) ;;; notice the spaces on each side of the quotes - we simply quoted a quote.
* Since differentiation is recursively defined, DER calls appropriate DERxxx functions which might then call DER
* 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.

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

|  |  |  |  |
| --- | --- | --- | --- |
| WFF | Derivative Function invoked by DER | Evaluated Expression in DER | Parameter Received by Derivative Function |
| (SIN X) | DERSIN | (DERSIN '(X)) | (X) |
| (SIN (G 3 X 2)) | DERSIN | (DERSIN '((G 3 X 2))) | ((G 3 X 2)) |
| (+ (G 3 X 2) 5) | DER+ | (DER+ '((G 3 X 2) 5)) | ((G 3 X 2) 5) |

**DERIVATIVE FORMULAS**

|  |  |  |
| --- | --- | --- |
| 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) | f(x)g'(x) + f'(x)g(x) |
| Chain Rule | f(g(x)) | f'(g(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(g(x)) | COS(g(x)) \* g'(x) |

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

;;; \*\*\*\*\*\* D E R U - \*\*\*\*\*\*

;;; (DERU- expr)

;;; expr - list around a single expression

;;; Note: derivative of (U- WFF) is (U- (der wff))

;;; Returns the derivative of the (U- WFF)

(defun derU- (expr)

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

)

Please download cs3723p4Test.txt for some helpful functions and test cases that must be used in your results.

**EXTRA CREDIT**

Several students who didn’t do well on the midterm exam requested extra credit work. That work must be available to anyone. 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.

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

Simplify resulting derivatives. Some examples:

|  |  |
| --- | --- |
| Resulting Derivative Without Simplification | Simplified |
| (+ (G 3 X 2) 0) | (G 3 X 2) |
| (+ 0 (G 3 X 2)) | (G 3 X 2) |
| (G 2 X 0) | 2 |
| (- (G 5 X 2) (G 5 X 2)) | 0 |
| (- 2 2) | 0 |
| (U- (U- (G 5 X 2))) | (G 5 X 2) |

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

**EXTRA CREDIT #2** (points = 5 + 100 / n where n is the number of people with FULL credit 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- (G 3 X 2)) (G 5 X 3))) | (\* (G 3 X 2) (G 5 X 3)) |
| (U- (\* (G 3 X 2) (U- (G 5 X 3)) )) | (\* (G 3 X 2) (G 5 X 3)) |
| (\* (U- (G 3 X 2)) (U- (G 5 X 3))) | (\* (G 3 X 2) (G 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.
  + 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