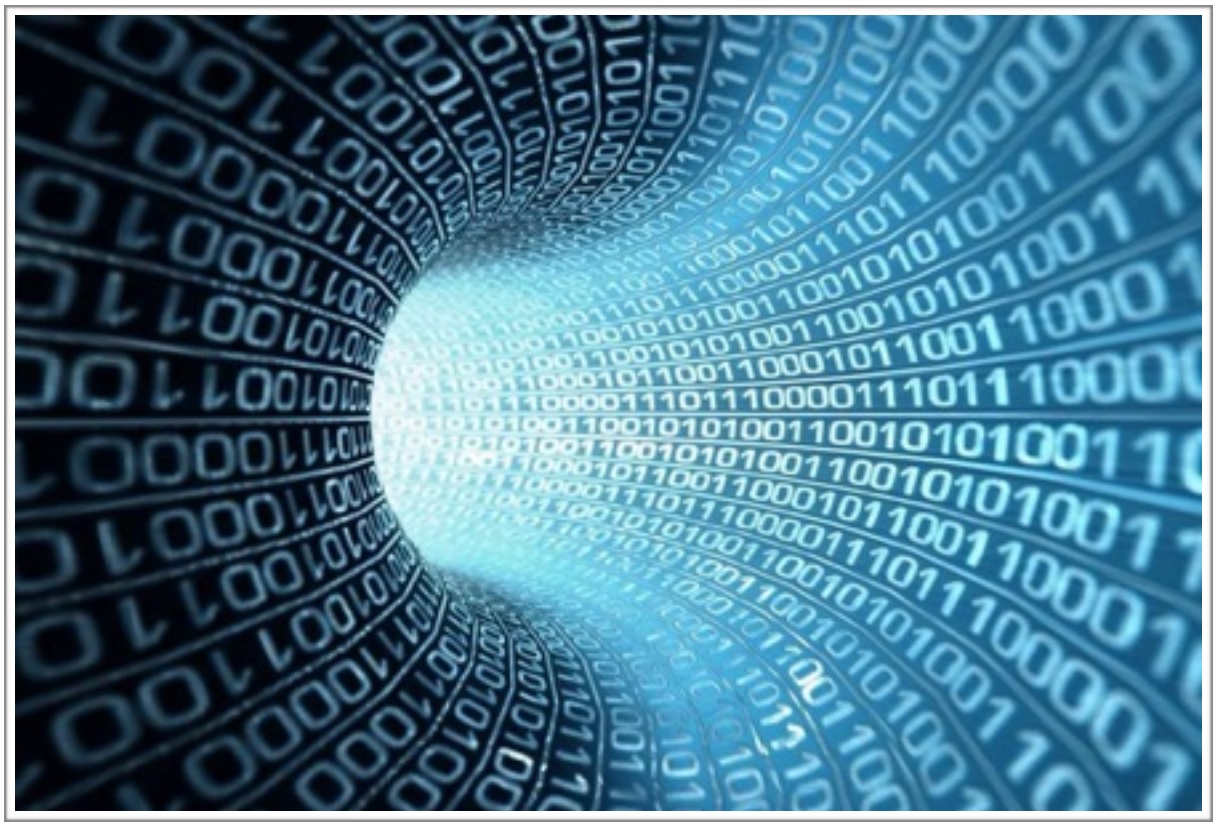
Proyecto Final: Derivador Analítico



Lenguajes de Programación

Jonathan Ginsburg
Pablo de la Mora
Rodrigo Solana

Mayo 12, 2016

Manual de Usuario

1. Abra una sesión de línea de comandos y diríjase al directorio adjunto llamado “Códigos Fuente”.
2. Ejecute el intérprete de SWI-Prolog.
3. Cargue la base de datos definida en el archivo “derivatives.pl” mediante el comando “[derivatives].”.
4. Búsque la derivada de un polinomio bien formado (ver definición en la sección de Descripción Técnica) P mediante el comando “simplifiedDerivative(P, D).”, para lo que SWI-Prolog unificará D con el polinomio bien formado que representa a la derivada de P respecto de x.

Ejemplo

Sesión de SWI-Prolog para búsqueda de:

$$\frac{\mathrm{d}}{\mathrm{d}x}\left[\frac{ex^{\pi}-x^{2}}{ex+\pi x}\right],$$

que es equivalente al polinomio bien formado:

```
[/, [-, [*, e, [^, x, pi]], [^, x, 2]], [+, [*, e, x], [*, pi, x]]]
```

```
Jonathans-MacBook-Pro:Códigos Fuente MacBook$ ls
derivatives.pl
Jonathans-MacBook-Pro:Códigos Fuente MacBook$ prolog
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 7.2.3)
Copyright (c) 1990-2015 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- [derivatives].
true.

?- simplifiedDerivative([/, [-, [*, 2.71, [^, x, 3.14]], [^, x, 2]], [+, [*, 2.71, x], [*
, 3.14, x]]], SD) ; true.
SD = [/, [-, [*, [+, [*|...], [...|...]], [-, [...|...]|...]], [*, 5.85, [-|...]]], [^, [
+, [*, 2.71|...], [*|...]], 2]] [write]
SD = [/, [-, [*, [+, [*, 2.71, x], [*, 3.14, x]], [-, [*, 2.71, [*, 3.14, [^, x, 2.14]]],
  [*, 2, x]]], [*, 5.85, [-, [*, 2.71, [^, x, 3.14]], [^, x, 2]]]], [^, [+, [*, 2.71, x],
[*, 3.14, x]], 2]] :
true.

?- halt.
Jonathans-MacBook-Pro:Códigos Fuente MacBook$ _
```

Nosotros escribimos el comando:

```
simplifiedDerivative([/, [-, [*, 2.71, [^, x, 3.14]], [^, x, 2]], [+, [*,
2.71, x], [*, 3.14, x]]], SD) ; true.
```

haciendo una aproximación a los valores trascendentes e y π a 2.71 y 3.14 respectivamente. Considerando $e + \pi = 5.85$, el resultado obtenido es la derivada:

```
[/, [-, [*, [+, [*, 2.71, x], [*, 3.14, x]], [-, [*, 2.71, [*, 3.14, [^, x,
2.14]]], [*, 2, x]]], [*, 5.85, [-, [*, 2.71, [^, x, 3.14]], [^, x, 2]]]],
[^, [+, [*, 2.71, x], [*, 3.14, x]], 2]]
```

Descripción Técnica

Para realizar la tarea impuesta se consideraron las listas que representan a los polinomios en x. A estas les llamamos polinomios bien formados y se definen como sigue:

Polinomios Bien Formados (PBF)

- El símbolo 'x' es un polinomio bien formado, que representa al polinomio algebráico en x formado por x únicamente;
- Si *a* es un número real también es un polinomio bien formado, que representa al polinomio en x de grado cero con valor constante *a*;
- Si p y q son polinomios bien formados y *a* es un número real; también son polinomios bien formados:
    - [p], el cual representa al mismo polinomio p;
    - [-, p, q], el cual representa la diferencia de polinomios p - q;
    - [^, p, *a*], el cual representa la potencia *a* del polinomio p;
    - [/, p, q], el cual representa el cociente de polinomios p/q;
- Si p1, …, pn son polinomios bien formados también lo son:
    - [+, p1, …, pn], el cual representa la suma de polinomios p1+ … + pn;
    - [*, p1, …,pn], el cual representa el producto de polinomios p1* … * pn;
- Nada más es un polinomio bien formado.

Ya definido el material de trabajo se procede a su operación. Aprovechando la recursividad de la derivación algebraica se realiza el derivado de PBFs mediante recursividad. Para los casos base se toman los casos base de nuestro lenguaje:

Dado un PBF p, se revisa:

- si es x, en cual caso se dice que su derivada es igual al número 1;
- si es un número real *a* en cual caso se se dice que su derivada es igual al número 0;

Luego se toman en cuenta los casos recursivos:

Dado un PBF p, se revisa:

- si p = [q] para algún PBF q, en cual caso se dice que su derivada es la derivada de q;
- si p = [-, q, r] para dos PBFs q y r, en cual caso se dice su derivada es [-, q', r'], donde q' y r' son las derivadas de los PBFs q y r, respectivamente;
- si p = [^, q, *a*] para un PBF q y run real *a*, en cual caso se dice que su derivada es [*, *a*, [^, q, *a*-1], q'], donde q' es la derivada del PBF q;

- si p = [/, q, r] para dos PBFs q y r, en cual caso se dice que su derivada es [/, [-, [*, q', r], [*, q, r']], [^, r, 2]], en donde q' y r' representan a las derivadas de los PBFs q y r, respectivamente;
- si p = [+, q1, …, qn] para n PBFs, en cual caso se dice que su derivada es [+, q1', …, qn'], donde qi' representa la derivada del iésimo PBF;
- si p = [*, q1, …, qn] para n PBFs, en cual caso se dice que su derivada es [+, [*, q1', [*, q2, …, qn]], [*, q1, [*, q2, …, qn]']], donde q1' es la derivada de q1 y [*, q2, …, qn]' es la derivada de [*, q2, …, qn].

Obsérvese que la operación de derivada es cerrada en los PBFs. Por lo tanto el proceso de simplificación también se aplica sobre PBFs.

La simplificación se realiza por pasos, repitiendo hasta que el proceso de simplificación no surta ningún cambio. También este proceso es cerrado bajo los PBFs, por tanto se puede operar la imágen de la operación tantas veces se quiera. La simplificación se realiza separando números y expresiones de x de PBFs. Así se opera con los números y literales, para lo que en caso de tener eliminación de término completo se regresa el neutro operativo del operador en cuestión. Por ejemplo: si se tiene [*, [-, x, x], x] se regresa [*, 1, x] y como hubo cambios se repite el proceso de simplificación y se regresa x y como hubo cambios se repite y se regresa x, lo cual no surtió efectos y se detiene el proceso.

Pseudocódigo

```
derivative(x, 1) if 1.
derivative(a, 0) if isNumber(a).
derivative([P], D) if derivative(P, D).
derivative([-, P, Q], D) if
     (derivative(P, DP) and
     derivative(Q, DQ) and
     D = [DP, DQ]).
derivative([^, P, a], D) if
     (derivative(P, DP) and
     D = [*, a, [^, P, a-1], DP]).
derivative([/, P, Q], D) if
     (derivative(P, DP) and
     derivative(Q, DQ) and
     D = [/, [-, [*, DP, Q], [*, DQ, P]], [^, Q, 2]]).
derivativ([+, P, Q], D) if
     (derivative(P, DP) and
     derivative(Q, DQ) and
     D = [+, DP, DQ]).
derivative([+, P|T], D) if
     (derivative(P, DP) and
     derivative([+|T], DT) and
     D = [+, P, DT]).
derivative([*, P, Q], D) if
     (derivative(P, DP) and
     derivative(Q, DQ) and
     D = [+, [*, DP, Q], [*, P, DQ]]).
derivative([*, P|T], D) if
     (derivative(P, DP) and
     derivative([+|T], DT) and
     D = [+, [*, DP, [+|T]], [*, DT, P]]).
```

```
simplify([X], X).
simplify(L, S) if
     simplifyEach(L, S).
simplify(P, S) if
     (isOP(P) and
     split(P, N, NN, OP) and
     reduce(N, OP, RN) and
     simplify(NN, SNN) and
     S = [OP, RN|SNN]).
```

Obsérvese que la relacion split(P, N, NN, OP) toma un PBF y lo separa en valores numéricos N, en no numéricos NN y en el operador OP. Por otro lado isOP(P) dice si una lista tiene un operador como primer elemento. Finalmente, reduce(N, OP, RN) aplica el operador OP a la lista de números N para producir RN.

Referencias

No se consultó ninguna fuente.