Pregunta 1

1. Chequear la imagen llamada ArbolSLD.png
2. La decisión de codificación que produce el error es en la cláusula de “descend(X,Y):- descend(Z,Y), child(X,Z).” porque vemos que hace la recursión primero y después busca el child que está directamente a él, sin embargo, como vemos en nuestro árbol SLD sólo se hace la parte recursiva, ya que a descend se le está pasando una variable Z que no es encontrada hasta que unifique con un hecho del child.
3. El código que yo propongo sería invertir esos dos hechos, o sea que primero se busque el child y después haga la recursión. El código sería el siguiente:

child(anna,bridget).

child(bridget,caroline).

child(caroline,donna).

child(donna,emily).

descend(X,Y):- child(X,Z), descend(Z,Y).

descend(X,Y):- child(X,Y).

Pregunta 2

1. El código sería el siguiente:

path(X, Y) :- connected(X, Y).

path(X, Y) :- connected(X, Z), path(Z, Y).

1. El código sería el siguiente:

path(X, Y, []) :- connected(X, Y).

path(X, Y, [H1|T1]) :- connected(X, Z), H1 = Z, path(Z, Y, T1).

1. Chequear la imagen ArbolSLD(2).png

Pregunta 3

1. Este código lo que hace es identificar si el número que se pasó como primer parámetro es positivo, negativo o cero. Esto se hace porque en la primera cláusula unifica si el número pasado en el primer argumento es mayor que cero, en la segunda cláusula sólo unifica si lo que se pasa como primer parámetro es cero y en la tercera cláusula unifica si el número pasado es menor que cero.
2. Este código es ineficiente porque vemos que, para casos diferentes de cero o iguales a cero, aunque ya el programa encontró su respuesta por unificación, ésta sigue por el otro camino que tiene. Esto lo vemos cuando probamos un query parecido a “class(100, X).”, que aquí vemos como nos imprime que X es igual a positive, pero además nos dice un false como el segundo camino que encontró.
3. Código:

class(Number, positive) :- Number > 0, !.

class(0, zero) :- !.

class(Number, negative) :- Number < 0.

En esta nueva versión podemos ver que hay cortes verdes porque estos se activarán siempre de que la primera o segunda cláusula sean la que unifique, pero una vez que esto pase sabemos que la respuesta ya fue encontrada y que no tiene que buscar otro camino, por lo tanto, así también evitamos el problema de ineficiencia del segundo punto. En caso de que no unifique en ninguna de estas cláusulas, entonces pasará a la última cláusula y si unifica, la respuesta será true o negative, pero si no es el caso el programa simplemente nos responderá con un false.