4. However, this keeps deconstructing my Accumulator

rev\_filter([], \_, Accumulator). %once the list is empty.

rev\_filter([E|Tail],E, Accumulator) :-

rev\_filter(Tail, E, Accumulator).

rev\_filter([Head|Tail],E,Accumulator) :-

not(Head = E),

rev\_filter(Tail, E, [Head|Accumulator]).

8a. base + 0 \* size

8b. base+1\*size

8c. base+(i\*n\*size)+(j\*size)

8d. base + (i\*size)+(j\*m\*size)+(k\*size)

9a. I would want to use gradfather2, since it checks male first, which is smaller than parent, so if male fails, we do not have to worry about checking for a parent.

9b. I would want to use grandfather1. Since parent has less facts, it would help end the program early if it returns false.\

9c1. GF1 not backtrack, GF2 does backtrack

parent(joe, bob).

parent(bob, alice).

male(bob).

male(joe).

When Grandfather1(X,Y) is called:  
1. Calls parent(X,Z), which goes to the first parent fact : parent(joe, bob) - true

2. check if X is male: male(joe) - true

3. checks parent(bob, Y) , which calls second parent fact: parent(bob, alice) – true

4. returns: grandfather1(joe, alice) – true

5. X = joe, Y = alice

When Grandfather2(X,Y) is called:

1. Calls male(X), which calls first male fact: male(bob) – true
2. calls parent(bob, Z), which calls the second parent fact: parent(bob, alice) – true
3. calls parent(alice, Y), which fails because there is no fact parent(alice, \_)
4. backtracks at call parent(alice,Y), and calls second male fact: male(joe) – true
5. calls parent(joe, Z), which calls the first parent fact: parent(joe, bob) – true
6. calls parent(bob, Y) which calls the second parent fact: parent(bob, alice).
7. Returns: grandfather1(joe, alice) – true
8. X = joe, Y = alice

9c2. GF1 backtracks, GF2 does not

parent(bob, alice).

parent(joe, bob).

male(joe).

male(bob).

For grandfather1(X,Y):

1. calls parent(X, Z), which goes to the first parent fact: parent(bob, alice) – true
2. checks if bob is male – true
3. calls parent(alice, Y), but fails because there is no fact parent(alice, \_),
   1. **must backtrack because of step 3**
4. calls the second parent fact, parent(joe, bob) – true
5. Checks if joe is male - true
6. Calls parent(bob, Y), which goes to our first parent fact: parent(bob, alice) - true
7. returns grandfather1(joe, alice) – true
8. X = joe, Y = alice

For grandfather2(X,Y):

1. Calls male(X), which goes to the first male fact: male(joe)
2. Calls parent(joe, Z), which goes to the second parent fact: parent(joe, bob) – true
3. Calls parent(bob, Y), which goes to the first parent fact: parent(bob, alice) – true
4. Returns grandfather2(joe, alice) – true
5. X = joe, Y = alice