# BASIC PROLOG

- Arithmetic using built-in operators
- Concatenating lists using *append* predicate
- Mapping example of transforming a list
- Reversing a List
- Syntax of data objects

# Arithmetic in Prolog

**For efficiency uses built-in operators e.g. X is 3 + 5, >, <**

**Example:**

```
%factorial( N, F) :- F is the integer N factorial
factorial( N, F) :- N > 0, N1 is N - 1,
                    factorial(N1, F1), F is N * F1.
factorial( 0, 1).
```

- **cannot be used with 1st argument as variable**

**Iteration using Recursion** factorial/4 = 4 arguments

**Example:**

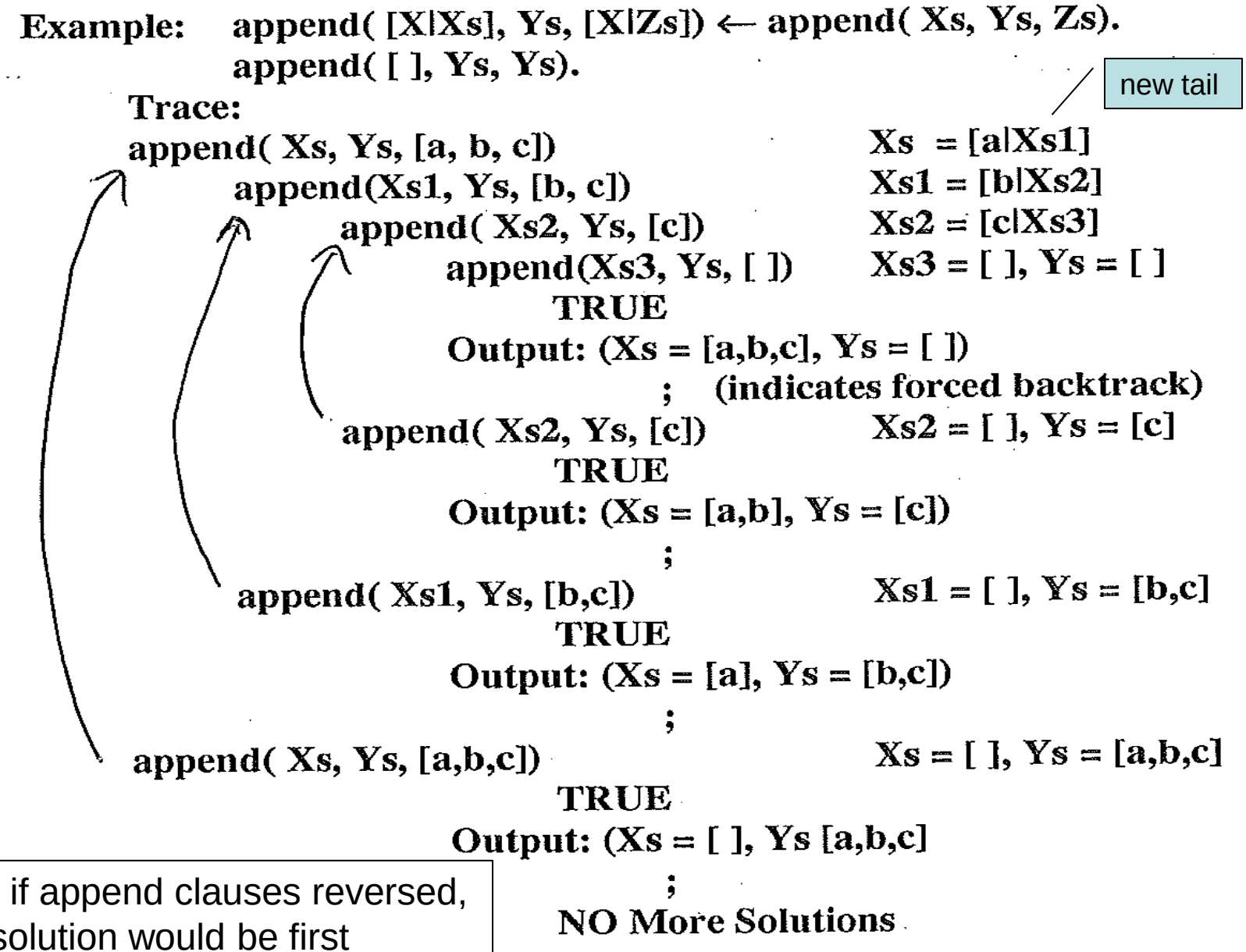
```
factorial( N, F) :- factorial( 0, N, 1, F).
factorial( I, N, T, F) :- I < N, I1 is I + 1,
                          T1 is T * I1, factorial( I1, N, T1, F).
factorial( N, N, F, F).
```

- **computes :**

```
I is 0; T is 1;
    WHILE I < N DO
        I is I + 1; T is T * I    END;
    RETURN T.
```

- **I and T are values of loop vars before (I + 1)th iteration**
- **I is loop counter and T the accumulator**
- **4th arg of factorial/4 is instantiated to the solution**

# Example Append with 3rd argument variable




Note if append clauses reversed, last solution would be first

# Example – mapping program

**Example of a mapping program:**

```
%mapping( Xs, Ys) :- Ys is the result of applying a mapping to Ys
mapping( [X|Xs], [Y|Ys]) :-
        map( X, Y), mapping( Xs, Ys).
mapping([ ], [ ]).
map('A', a).
map('B', b).
map('C', c).
map('D', d).
```

Quotation ‘ ’ indicates constant

? mapping([‘B’, ‘D’], Z).

1st recursion Y=b
2nd recursion Y=d

# Example Reverse/2 = 2 arguments

```
%reverse( List, RevList)  :-  RevList is the result of reversing List
reverse( [ ], [ ] ).
reverse( [ X | Xs ], Zs ) :-   reverse( Xs, Ys),
        append( Ys,[ X ], Zs).
```

reverse([1],R)  gives  append([],[1],R)

reverse([2,1],R) gives append([1],[2],R)

reverse([3,2,1],R) gives append([1, 2],[3],R)

# Example Reverse/3 = 3 arguments

An extra argument can be used to build bottom-up:

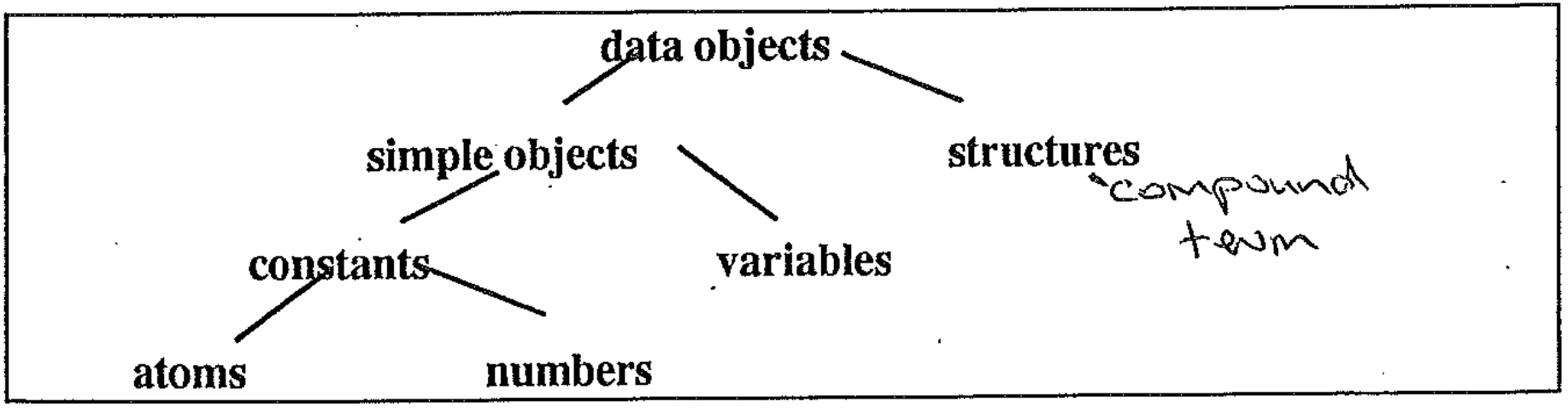
```
%reverse( List, Tsil) :- Tsil is the result of reversing List
reverse( Xs, Ys) :- reverse( Xs, [ ], Ys).
reverse( [X|Xs], Revs, Ys) :- reverse( Xs, [X|Revs], Ys).
reverse( [ ], Ys, Ys).
```

reverse/3 is introduced with 2nd arg as an accumulator

```
Trace:      reverse([a,b,c],Xs)
                reverse([a,b,c], [ ], Xs)
                    reverse([b,c], [a], Xs)
                        reverse([c], [b,a], Xs)
                            reverse([ ], [c,b,a], Xs)
                                Xs = [c,b,a]
                            TRUE
```

**Note that 3rd argument is carried through recursion, and instantiated to reversed list**

# Data Object terminology




- atoms are strings of characters from upper-case, lower-case, digits, special chars and can be strings of:
  1. letters, digits and _, starting with lower-case
  2. special characters e.g. + - * / < > = & _
  3. strings in single quotes

- variables are strings of upper-case, lower-case, digits, _ starting with upper-case or _
  - a single _ represents the anonymous variable