

## Programming Languages and Programming Methodologies

Some notes on recursion.

PLPM 2007-2008

### Known recursive definitions

- One of my **ancestors** is  
one of my parents  
or one of their **ancestors**.
- A **string** of characters is  
a single character  
or a single character followed by a **string** of  
characters.
- To **decouple** a train  
uncouple the first carriage  
and then **decouple** the rest of the train.

PLPM 2007-2008

### To do

- Identify the base case and the recursive case.
- For the recursive case, indicate how the **problem reduction** is realised (i.e how the problem is solved in terms of **simpler** problems).

PLPM 2007-2008

### Example Recursive Program

- You talk about someone either if you know him or if you know someone who talks about him.
- % A talks about B  
talks\_about(A,B) :- knows(A,B).  
talks\_about(P,R) :- knows(P,Q), talks\_about(Q,R).  
knows(bill,jane).  
knows(jane,pat).  
knows(jane,fred).  
knows(fred,bill).  
?- talks\_about(X,Y).

PLPM 2007-2008

## List Processing

- Lists are recursive data structures.
- List processing: number of different ways.
- Program patterns: schemata that can be used as a source of inspiration:
  - test for existence
  - test all elements
  - return a result - having processed one element
  - return a result - having processed all elements

PLPM 2007-2008

## Test for Existence

- Aim: we want to determine that some collection of objects has at least one object with a desired property.
- Example: that a list of terms has at least a term which is also a list.

- Schema:

```
list_existence_test(Info, ( Head | Tail)) :-  
    element_has_property(Info, Head).  
list_existence_test(Info, ( Head | Tail)) :-  
    list_existence_test(Info, Tail).  
% Info: parameters that carry around information
```

PLPM 2007-2008

## Example 1: Test for Existence

- Example: that a list of terms has at least a term which is an integer.

- 

```
listwithinteger(( Head | Tail)) :-  
    integer( Head).  
listwithinteger(( Head | Tail)) :-  
    listwithinteger(Tail).
```

PLPM 2007-2008

## Example 2 : Test for Existence

- Does member/2 fit into this pattern, when it is used with two given arguments?
- What is then the element\_has\_property test?

PLPM 2007-2008

## Test All Elements

- Aim: we require that all the objects in the collection (i.e. all the elements of the list) satisfy some property.
- Example: all elements in the list are digits.
- Schema:

```
test_all_have_property(Info, ()).
test_all_have_property(Info, (Head | Tail)) :-
    element_has_property(Info, Head),
    test_all_have_property(Info, Tail).
```

PLPM 2007-2008

## Example: Test All Elements

- Example: all elements in the list are digits.
- Schema:

```
all_digits( ( ) ).
all_digits( ( Head | Tail ) ) :-
    member( Head, (0,1,2,3,4,5,6,7,8,9)),
    all_digits( Tail ).
```

PLPM 2007-2008

## Return a Result Having Processed One Element

- Also a **result** argument
- Aim: work through a list until an element satisfies some condition whereupon we stop and return some result.
- Example: select the part of the list after any occurrence of the element a.
- Schema:

```
return_after_event(Info, (Head | Tail), Result) :-
    property(Info, Head),
    construct_result(Info, Head, Tail, Result),
    return_after_event(Info, (Head | Tail), Result) :-
    return_after_event(Info, Tail, Result).
```

PLPM 2007-2008

## Example: Return a Result Having Processed One Element

- Example: select the part of the list after any occurrence of the element a.

```
everything_after_a( ( Head | Tail ), Result ) :-
    Head = a,
    Result = Tail.
everything_after_a( ( Head | Tail ), Result ) :-
    everything_after_a( Tail, Result ).
?- everything_after_a( (d,s,a,b,n,m), Tail ).
    Tail = (b,n,m)
?- everything_after_a( (d,s,a,b,a,c), J ).
    J = (b,a,c).
;
J = (c)
```

PLPM 2007-2008

## Return a Result Having Processed All Elements

- Aim: work through a list and transform each element into a new element (this can be seen as a mapping).
- Example: a program that takes a list of integers and 'triples' each of them.

- *Schema:*

```
process_all(Info, [], 0).
process_all(Info, (H1 | T1), (H2 | T2)) :-
    process_one(Info, H1, H2),
    process_all(Info, T1, T2).
```

PLPM 2007-2008

## Example 1 : Return a Result Having Processed All Elements

- Example: a program that takes a list of integers and 'triples' each of them.

- 

```
triple([], 0).
triple((H1 | T1), (H2 | T2)) :-
    H2 is 3 * H1,
    triple(T1, T2).
```

- *Question: is the recursive call 'simpler' than the head of the recursive clause???*

PLPM 2007-2008

## Example 2: Return a Result

- Consider the predicate `nth(N,L,EI)` which for a given positive integer `N` and a given list `L`, returns the `N`th element in the list `L`, as `EI`.
- Under which schema does it fall?
- Write it.

PLPM 2007-2008

## Example 3: Return a Result

- Common error in constructing lists!!!!

```
triple([], []).
triple((H1 | T1), T2) :-
    H2 is 3 * H1,
    triple(T1, (H2 | T2)).
?- triple([1, 2, 3], A).           % use recursive clause
?- triple([2, 3], (3 | A)).
?- triple([3], (6, 3 | A)).
?- triple([], (9, 6, 3 | A)).    % use base clause
failing unification: () = (9, 6, 3 | A)
```

observation: complexity of the recursive calls???  
no way to pass back "the computed result" to the query

PLPM 2007-2008

## Two Approaches to Return Results

- **Building structures in the clause head**

this is the same as return a result - having processed all arguments

```
triple(0, 0).  
triple((H1 | T1), (H2 | T2)) :-  
  H2 is 3 * H1,  
  triple(T1, T2).
```

*the output structure consists of two parts: a bit we can calculate easily (H2) and another bit which requires a recursive call (T2)*

PLPM 2007-2008

## Two Approaches to Return Results

- **Building structures in the clause body**  
this uses an **accumulator** (compare with wrong triple!!)

```
triple(0, Y, Y).  
triple((H1 | T1), X, Y) :-  
  H2 is 3 * H1,  
  triple(T1, (H2 | X), Y).
```

*the recursive call is simpler in the first argument (the recursion argument) and more complex in the second argument (the accumulator)*

*does the result argument get changed???*

PLPM 2007-2008

## Two Approaches to Return Results

- **Building structures in the clause body**  
this uses an **accumulator** (compare with wrong triple!!)
- Schema:

```
process_all(Info, [], AccAcc).  
process_all(Info, (H1 | T1), Acc, Ans) :-  
  process_one(Info, H1, H2),  
  process_all(Info, T1, (H2 | Acc), Ans).
```

PLPM 2007-2008

## To do

- Build proof trees for the triple/2 and triple/3 queries....

PLPM 2007-2008