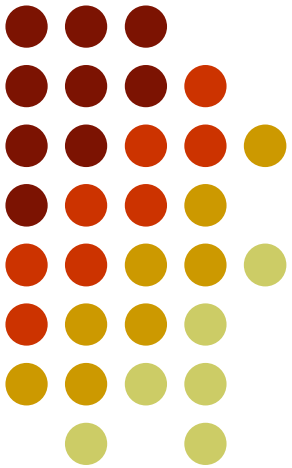
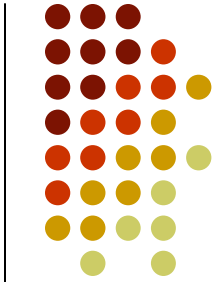
Introduction to Prolog Accumulators, Debugging

CS181: Programming Languages

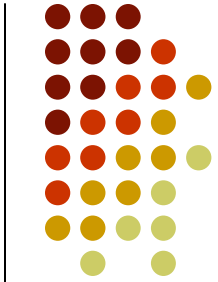


Topics:



- More list processing
- Accumulators
- Prolog in action: Towers of Hanoi
- Debugging

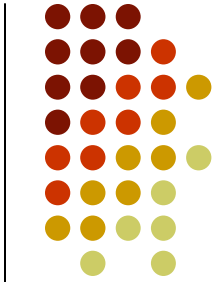
List processing



- The goal nextto(X,Y,L) succeeds if elements X and Y are consecutive elements of list L:

```
nextto(X, Y, [X,Y|_ ]).
nextto(X, Y, [ _|Z]) :- nextto(X, Y, Z).
```

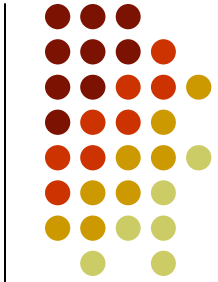
List processing



- Delete all occurrences of element X from list L1 and create the resulting list L2:

```
delete(_, [ ], [ ]).
delete(X, [X|L], M) :- !, delete(X,L,M).
delete(X, [Y|L1], [Y|L2]) :- delete(X, L1, L2).
```

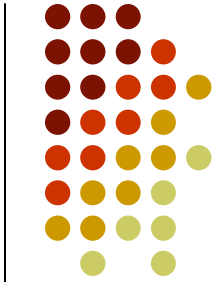
List processing



- The rule subst(X,L,A,M) constructs a new list M made up from elements of list L, except that any occurrence of X is replaced by A:

```
subst(_, [ ], _, [ ]).
subst(X, [X|L], A, [A|M]) :- !, subst(X, L, A, M).
subst(X, [Y|L], A, [Y|M]) :- subst(X, L, A, M).
```

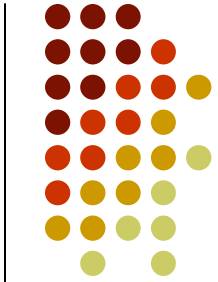
List processing



- The rule sublist(X,Y) suceeds if X is a sublist of Y, that is, if every element of X appears in Y, consecutively, in the same order:

```
sublist([X|L], [X|M]) :- prefix(L, M), !.
sublist(L, [ _|M]) :- sublist(L, M).
```

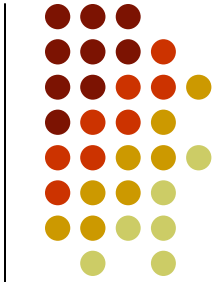
Accumulators



- An accumulator is an argument of the predicate used to represent the “answer so far”.

```
len(L, N) :- lenacc(L, 0, N).

lenacc([ ], A, A).
lenacc([H|T], A, N) :- A1 is A + 1,
                       lenacc(T, A1,N).
```

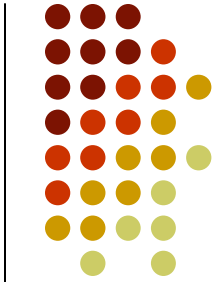
Accumulators



[a, b, c, d, e]

lenacc([a, b, c, d, e], 0, N)
lenacc([b, c, d, e], 1, N)
lenacc([c, d, e], 2, N)
lenacc([d, e], 3, N)
lenacc([e], 4, N)
lenacc([ ], 5, N)

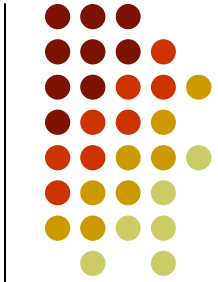
Accumulators



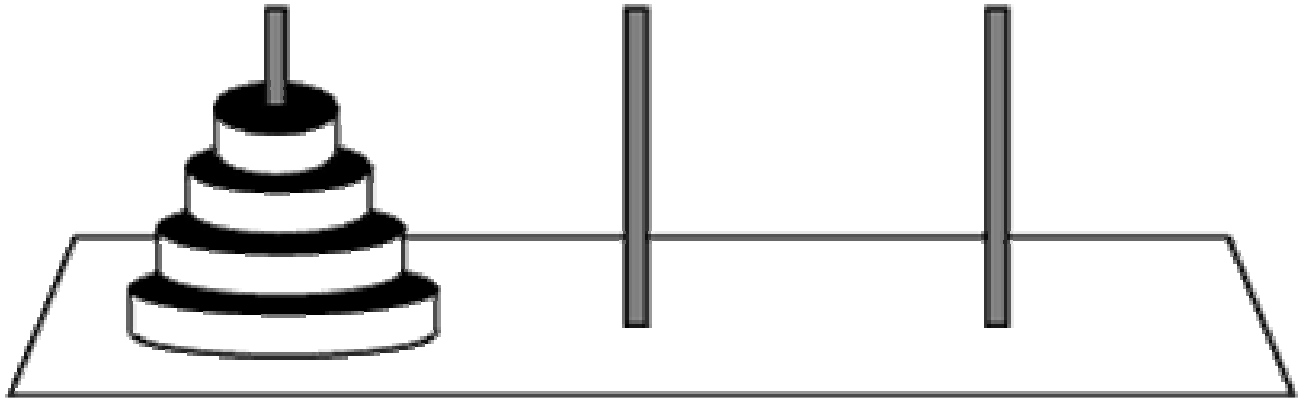
- The rule remdup(L,M) succeeds if M is a list with same elements as L, but the duplicate elements are removed:

```
remdup(L, M) :- dupacc(L, [ ], M).

dupacc([ ], A, A).
dupacc([H|T], A, L) :- member(H, A),
                       dupacc(T, A, L).
dupac([H|T], A, L) :- dupacc(T, [H|A], L).
```

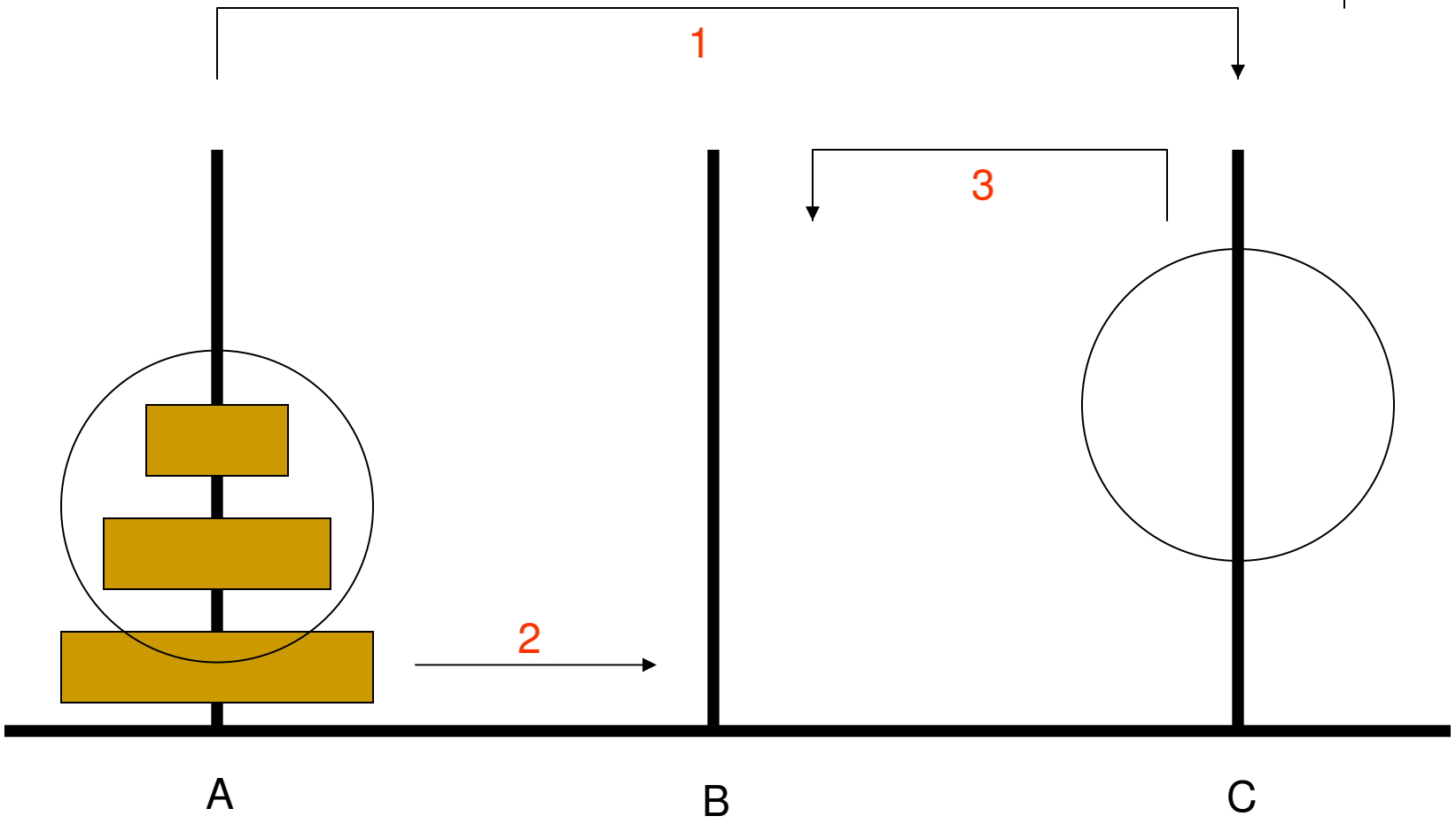
Towers of Hanoi



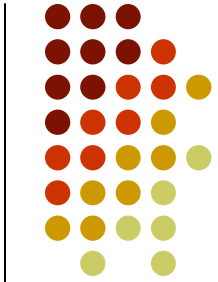
- Stack of *n* disks arranged from largest on the bottom to smallest on top placed on a rod
- Two empty rods: goal and an auxiliary rod
- Minimum number of moves to move the stack from one rod



Towers of Hanoi

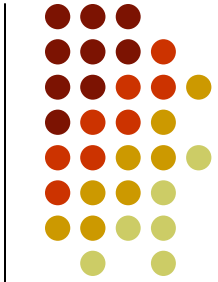


Towers of Hanoi



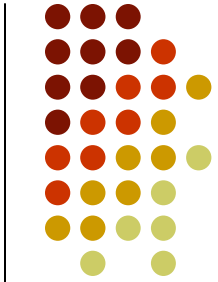
```
hanoi(N) :- move(N, left, centre, right).

move(0, _, _, _) :- !.
move(N, A, B, C) :- M is N-1,
     move(M, A, C, B),          % 1
     inform(A, B),              % 2
     move(M, C, B, A).          % 3
inform(X,Y) :-
     write([move, disk, from, X,  to, Y]), nl.
```

Debugging



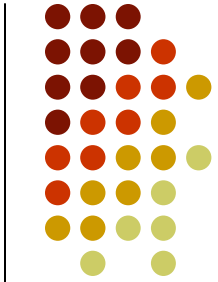
- Don’t forget to write the dot .
- Add at least one “white space” after the dot
- Some characters belong in pairs: ( ), [ ], /**/
- Do not misspell names of facts, rules, built-in predicates
- Use parentheses to define explicitly the associativity of operators.

Debugging (lists)



- How do [a,b,c] and [X|Y] match?
- Do [a] and [X|Y] match?
- Do [ ] and [X|Y] match?
- Is [X, Y | Z] meaningful?
- Is [X | Y, Z] meaningful?
- Is [X | [Y | Z]] meaningful?
- Is there more than one way to match to lists?

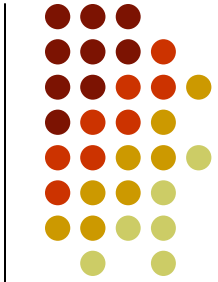
Debugging (Tracing Model)



- The trace predicate prints out information about the sequence of goals in order to show where the program has reached in its execution

- Example (see trace_example.pl)

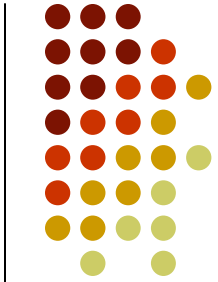
Debugging (Tracing Model)



Some of the events which may happen during a trace:

- CALL: A CALL event occurs when Prolog tries to satisfy a goal
- EXIT: An EXIT event occurs when some goal has just been satisfied
- REDO: A REDO event occurs when the system comes back to a goal, trying o re-satisfy it
- FAIL: A FAIL event occurs when a goal fails

References



- Clocksin, W.F., and Mellish C.S. *Programming in Prolog*. 4th edition. New York: Springer-Verlag. 1994.
- Van Le, T. *Techniques of Prolog Programming*. John Wiley & Sons, Inc. 1993.