PROLOG PROGRAMS AND SEARCH TREE

Comp3031 Lab 10
Fall 2019

Chenyang LEI& Lipeng WANG

Equality & Unification

- Equality test:
  - Term comparison: == \==
  - Arithmetic comparison: =:= =\=
  - These predicates do not unify the operands.
- Unification:
  - E1 = E2 returns true if E1 and E2 can be unified.
  - ?- X = a.

    X = a.
  - ?- X is 3.

    X = 3.

Unification

- Prolog answers a query by proof search of goals.
- Prolog searches the database by deciding whether a goal unifies with the fact or the head of a rule.
  - An uninstantiated variable will unify with any object.
  - An atom or number will unify only with itself.
  - A structure will unify with another structure if
    - They have the same functor and number of arguments.
    - All the corresponding arguments can unify.

Unification Examples

- ?-a = a.
- ?-a = b.
- ?-X = a.
- ?-foo(a,Y) = foo(X,b).
- ?-2*3+4 = X + Y.
- ?-[a,b,c] = [X,Y,Z].
- ?-[a,b,c] = [X | Y].

Prolog Search Tree

- A tree represents the search process of Prolog.
- If a node N1 is a child of the node N2, then the problem of proving the goal for N2 can be solved by (reducing to) proving the goal for N1.

Prolog Search Tree (cont.)

- The empty goal means nothing to prove, thus is ”succeeded”.
- A leaf, which is a node without children, with non-empty goal is a dead-end: there is no way to prove the goal, and is thus ”failed”.

Example

- Suppose we have the following database:

  f(a).

  f(b).

  g(a).

  g(b).

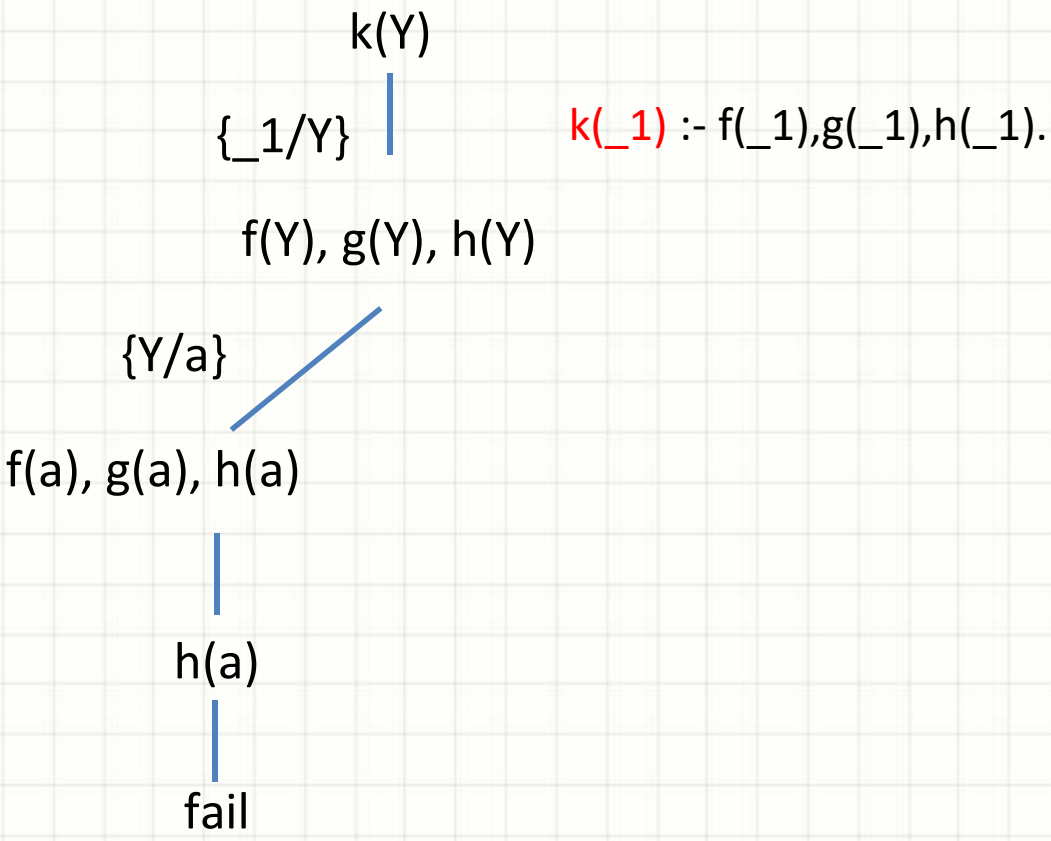
  h(b).

  k(X) :- f(X),g(X),h(X).

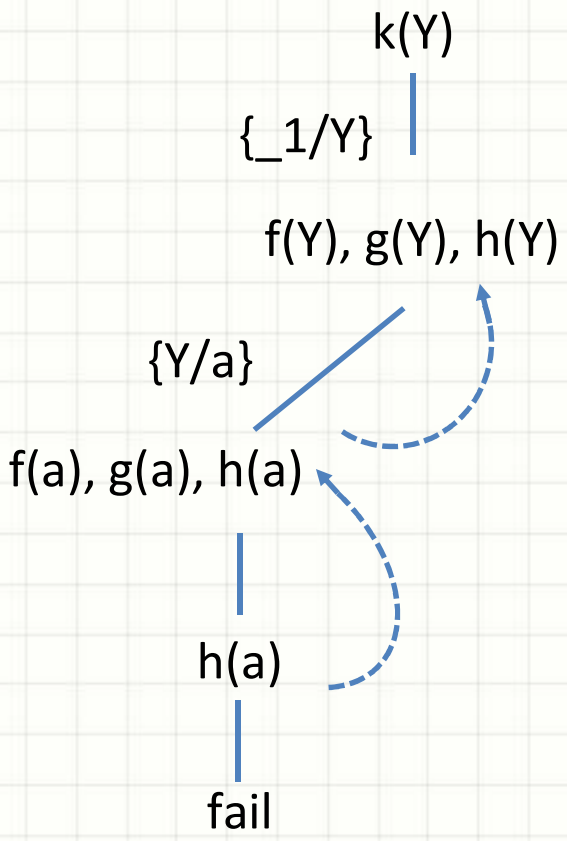
- Query:

  ?- k(Y).

Search Tree

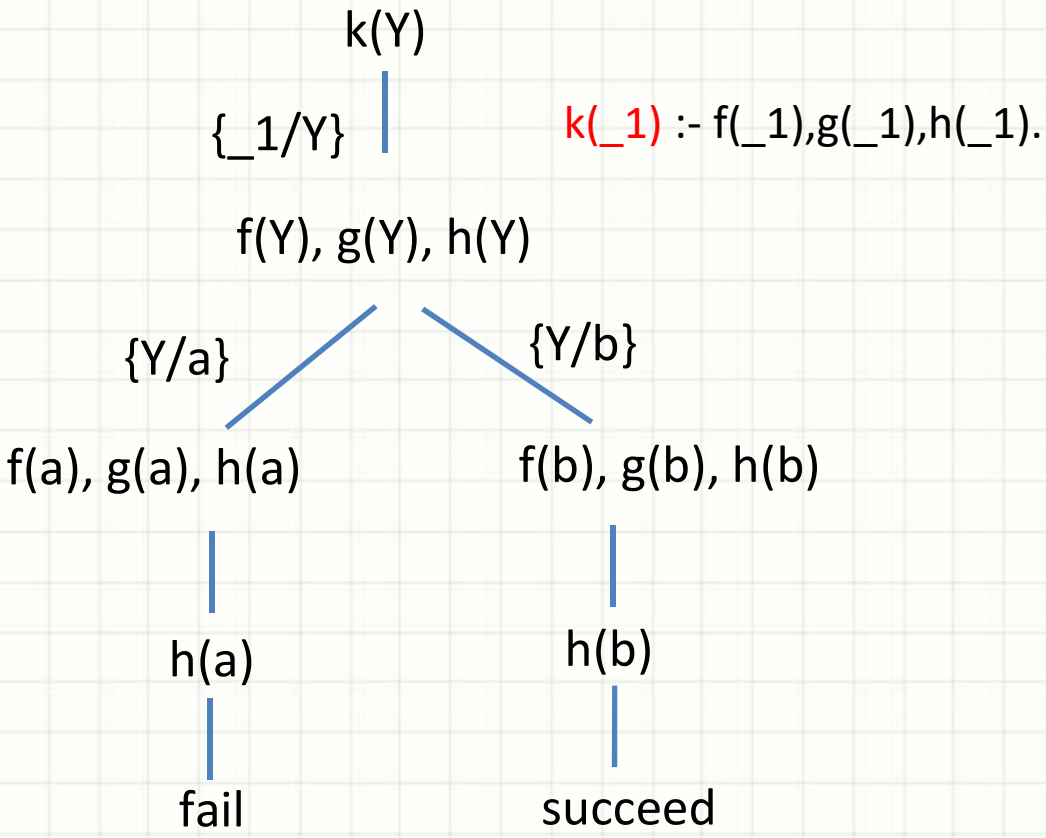


Search Tree



k(_1) :- f(_1),g(_1),h(_1).

Search Tree



Trace Mode

- Search process can be checked using Trace mode
  - ?- trace.
  - true
- Hitting return will show you the next step

Trace Mode (cont.)

- Call: Prolog tells what is the goal.
- Fail: the specified goal failed.
- Exit: the goal succeeded.
- Redo: Prolog is trying to find an alternative way of proving the goal.

Trace Example

- trace query

```
[trace]  ?- k(Y).
   Call: (6) k(_G2757) ? creep
   Call: (7) f(_G2757) ? creep
   Exit: (7) f(a) ? creep
   Call: (7) g(a) ? creep
   Exit: (7) g(a) ? creep
   Call: (7) h(a) ? creep
   Fail: (7) h(a) ? creep
   Redo: (7) f(_G2757) ? creep
   Exit: (7) f(b) ? creep
   Call: (7) g(b) ? creep
   Exit: (7) g(b) ? creep
   Call: (7) h(b) ? creep
   Exit: (7) h(b) ? creep
   Exit: (6) k(b) ? creep
Y = b.
```

Goal Order Matters

- Example:
  - person(X) :- person(Y), mother(Y, X).
    person(m).
    mother(m, j).
  - person(X) :- mother(Y,X), person(Y).
    person(m).
    mother(m, j).
  - person(m).
    mother(m, j).
    person(X) :- person(Y), mother(Y, X).
  - Query:
    ?- person(X).

Exercise Example

- Define a relation **count(X,L,N)** where N is the number of occurrences of X in L.
- Answer:
  - % base case
  - count(_,[],0).
  - % inductive case
  - count(X, [X|L], N) :- count(X, L, N1), N is N1+1.
  - count(X,[Y|L], N) :- X\==Y, count(X,L,N).
  - %query
  - count(5,[1,4,5,5,5],N).

Exercise1

- Given the **append** relation below:
  - append([], L, L).
  - append([H|T], L, [H|L1]) :- append(T, L, L1).
- Use **append(X,L1,L2)** to define **list_reverse(L1,L2)** where L2 is the reverse of L1.
- Example:
  - ?- list_reverse([7,up,8,down], L).
  - L = [down, 8, up, 7].

Exercise2

- Write a Prolog relation **list_prefix(L1,L)** to generate all the prefixes of L:
  - L1 is a sublist of L if and only if all elements of L1 appear consecutively in the same order as in L.
  - L1 is a prefix of L if and only if L1 is a sublist of L and the first element of L1 is the first element of L.

Exercise2

- Examples:
  - ?- list_prefix(X,[1,2,3]).
  - X = [];
  - X = [1];
  - X = [1,2];
  - X = [1,2,3];