Reading 22: A Third Look at Prolog

# Exercise 1: Summarize

Prolog excels in problem space search and numeric computation, using evaluable predicates and instantiation requirements.

# Exercise 2: Demonstrate & Explain

knapsackDecision(Pantry, Capacity, Goal, Knapsack) :-

subseq(Knapsack, Pantry), % Generate all possible subsequences of Pantry

weight(Knapsack, Weight), % Calculate the weight of each subsequence

Weight =< Capacity, % Filter 1: Check if weight is at or below capacity (4)

calories(Knapsack, Calories), % Calculate the calories of each subsequence

Calories >= Goal. % Filter 2: Ensure calories meet or exceed the goal

% Example query:

% ?- knapsackDecision([food(bread,4,9200), food(pasta,2,4500), food(peanutButter,1,6700), food(babyFood,3,6900)], 4, 10000, X).

% X = [food(pasta, 2, 4500), food(peanutButter, 1, 6700)].

% Stops at the first subsequence that is <= 4 kg and >= 10000 calories.

legalKnapsack(Pantry, Capacity, Knapsack) :-

subseq(Knapsack, Pantry),

weight(Knapsack, W),

W =< Capacity.

knapsackOptimization(Pantry, Capacity, Knapsack) :-

findall(K, legalKnapsack(Pantry, Capacity, K), L), % Generate all subsequences that meet the weight constraint and collect them into list L

maxCalories(L, Knapsack). % Find the subsequence with the maximum calories

% Example query:

% ?- knapsackOptimization([food(bread,4,9200), food(pasta,2,4500), food(peanutButter,1,6700), food(babyFood,3,6900)], 4, Knapsack).

% Knapsack = [food(peanutButter, 1, 6700), food(babyFood, 3, 6900)].

% Generates all subsequences and checks if their weight is valid before recording the full list. Picks the highest calories out of the complete filtered list.

Both the knapsackDecision (kDecision) and knapsackOptimization (kOptimize) predicates aim to find the highest calorie combination of foods while staying below a specified weight limit. In terms of similarities, both predicates start by generating all possible subsequences of the list of foods and then apply a weight limit filter. However, the differences between them become apparent immediately. kOptimize filters the subsequences by weight while generating the list, resulting in a clean list of valid subsequences from which it can calculate the total calories. On the other hand, kDecision is a short-circuit predicate that checks each subsequence in its list to find one that meets both the weight and calorie constraints, stopping as soon as it finds a match, regardless of whether it is the optimal solution. In contrast, kOptimize works with a pre-filtered list of subsequences and uses a predicate to select the subsequence with the highest total calories, ensuring the optimal solution is found.

# Exercise 3: Inquire

Which programming languages automatically evaluate and unify an arithmetic expression with its result? Explain why these languages behave this way.