TDT4165 - Programming Languages

Assignment 5: Relational and Constraint Programming

Benjamin Zubača

November 11, 2024

Task 1: Constraint Programming

```prolog
payment(Sum, Coins) :-
    maplist(coin_constraint, Coins),
    calculate_sum(Coins, Sum).

coin_constraint(coin(AmountNeeded, _, AmountAvailable)) :-
    AmountNeeded in 0..AmountAvailable.

calculate_sum([], 0).
calculate_sum([coin(AmountNeeded, Value, _) | Tail], Sum) :-
    calculate_sum(Tail, TailSum),
    Sum #= AmountNeeded * Value + TailSum.
```

The `payment/2` predicate ensures that the constraints for each coin in the list of `Coins` are met and then calculates the total `Sum` based on the coins used. The `coin_constraint/1` predicate enforces that `AmountNeeded` (the number of each coin required) is within the available range, defined by `AmountAvailable`. The `calculate_sum/2` predicate recursively calculates the `Sum` by iterating over each coin in the list, multiplying the `AmountNeeded` by its `Value`, and adding up the results.

Task 2: Relational Programming

Task 2.1: Create a Planner

```prolog
plan(StartCabin, EndCabin, Path, TotalDistance) :-
    explore(StartCabin, EndCabin, [StartCabin], 0, ReversedPath, TotalDistance),
    reverse(ReversedPath, Path).

explore(CurrentCabin, CurrentCabin, VisitedPath, AccumulatedDistance, VisitedPath, AccumulatedDistance).
explore(CurrentCabin, EndCabin, VisitedPath, AccumulatedDistance, FinalPath, TotalDistance) :-
    distance(CurrentCabin, NextCabin, StepDistance, 1),
    \+ member(NextCabin, VisitedPath),
    NewAccumulatedDistance is AccumulatedDistance + StepDistance,
    explore(NextCabin, EndCabin, [NextCabin | VisitedPath], NewAccumulatedDistance, FinalPath, TotalDistance).
```

The `plan/4` predicate finds a possible path between the two cabins, `StartCabin` and `EndCabin`, along with the total distance of that path. To accomplish this, it calls a helper predicate `explore/6`, which performs a recursive depth-first traversal of the graph. Starting from the `StartCabin`, `explore/6` builds the path by visiting each connected cabin one at a time, adding the next cabin to the list of visited cabins (stored in `VisitedPath`) and accumulating the distance traveled so far (`AccumulatedDistance`). If the end cabin is reached, `explore/6` returns the completed path and the accumulated distance. Since the path is constructed by prepending each cabin to the list, it ends up in reverse order. Therefore, the `reverse/2` predicate is used to correct the order before returning the final `Path` and `TotalDistance`.

Task 2.2: Create a Planner for the Shortest Path

```prolog
bestplan(StartCabin, EndCabin, Path, Distance) :-
    findall((D, P), plan(StartCabin, EndCabin, P, D), Paths),
    sort(Paths, [(Distance, Path) | _]).
```

The `bestplan/4` predicate finds the shortest possible path between two cabins, `StartCabin` and `EndCabin`, and its total distance. It does this by calling `plan/4` for every possible path between the two cabins and storing each resulting path and distance in a list. This list is then sorted by distance in ascending order. Since the shortest path will be the first element on the sorted list, `bestplan/4` extracts this path and its distance and returns them as the result.