Aula Prática 4

Modificação de Programas, Pesquisa

Objetivos:
- Alteração de Programas em tempo de execução
- Pesquisa em Grafos
- Jogos e Puzzles

1. Relações familiares revisitadas

Considere o exercício 1 da primeira ficha de exercícios, sobre relações familiares.

a) Implemente o predicado *add_person/0*, que questiona o utilizador sobre uma nova pessoa (male vs female, e nome), e acrescenta-a à base de conhecimento.

b) Implemente o predicado *add_parents(+Person)*, que questiona o utilizador sobre os progenitores de *Person*, acrescentando depois essa informação à base de conhecimento.

c) Implemente o predicado *remove_person/0* que questiona o utilizador sobre uma pessoa a remover da base de conhecimento, removendo-o, e a quaisquer relações em que a pessoa esteja envolvida.

2. Voos

Considere o seguinte excerto de código representativo de voos existentes:

```
%flight(origin, destination, company, code, hour, duration)
flight(porto, lisbon, tap, tp1949, 1615, 60).
flight(lisbon, madrid, tap, tp1018, 1805, 75).
flight(lisbon, paris, tap, tp440, 1810, 150).
flight(lisbon, london, tap, tp1366, 1955, 165).
flight(london, lisbon, tap, tp1361, 1630, 160).
flight(porto, madrid, iberia, ib3095, 1640, 80).
flight(madrid, porto, iberia, ib3094, 1545, 80).
flight(madrid, lisbon, iberia, ib3106, 1945, 80).
flight(madrid, paris, iberia, ib3444, 1640, 125).
flight(madrid, london, iberia, ib3166, 1550, 145).
flight(london, madrid, iberia, ib3163, 1030, 140).
flight(porto, frankfurt, lufthansa, lh1177, 1230, 165).
```

a) Implemente o predicado *get_all_nodes(-ListOfAirports)* que devolve uma lista com todos os aeroportos servidos pelos voos na base de dados, sem duplicados.

b) Implemente o predicado *most_diversified(-Company)* que devolve em *Company* a operadora com um número de destinos mais diversificado.

c) Implemente o predicado *find_flights(+Origin, +Destination, -Flights)*, que devolve em *Flights* uma lista com um ou mais (códigos de) voos que permitam ligar *Origin* a *Destination*. Use pesquisa em profundidade, evitando ciclos.

d) Implemente o predicado *find_flights_breadth(+Origin, +Destination, -Flights)*, com o mesmo significado que o predicado anterior, mas usando pesquisa em largura.

e) Implemente o predicado *find_all_flights (+Origin, +Destination, -ListOfFlights)*, que devolve em *ListOfFlights* uma lista com todas as possíveis formas de ligar *Origin* a *Destination* (cada uma representada como uma lista de [códigos de] voos).

f) Implemente o predicado *find_flights_least_stops(+Origin, +Destination, -ListOfFlights)*, que devolve em *ListOfFlights* uma lista com possíveis formas de ligar *Origin* a *Destination* (cada uma representada como uma lista de [códigos de] voos) com um mínimo de escalas, ie, uma lista com os caminhos mais curtos entre *Origin* e *Destination*.

g) Implemente o predicado *find_flights_stops(+Origin, +Destination, +Stops, -ListFlights)*, que devolve em *ListFlights* uma lista com possíveis formas de ligar *Origin* a *Destination* (cada uma representada como uma lista de [códigos de] voos) passando pelas cidades indicadas em *Stops*.

h) Implemente o predicado *find_circular_trip (+MaxSize, +Origin, -Cycle)*, que devolve em *Cycle* uma lista, de comprimento máximo *MaxSize*, com voos que iniciem e terminem em *Origin*, formando um ciclo.

i) Implemente o predicado *find_circular_trips(+MaxSize, +Origin, -Cycles)*, que devolve em *Cycles* uma lista de ciclos de tamanho máximo *MaxSize*, iniciando e terminando em *Origin*.

j) Um grafo ou subgrafo é fortemente conexo se a partir de qualquer um dos seus vértices se conseguir alcançar qualquer um dos outros vértices (direta ou indiretamente). Implemente o predicado *strongly_connected(+ListOfNodes)* que sucede se o conjunto de nós recebido como argumento (que pertence ao grafo de voos) constitui um subgrafo fortemente conexo, falhando caso contrário.

k) Implemente o predicado strongly_connected_components(-Components) que determina as componentes fortemente conexas maximais (não é possível acrescentar um nó/ramo sem que a componente deixe de ser fortemente conexa) do grafo de voos.

l) Uma ponte é uma aresta que não está contida em nenhum ciclo de um grafo. Implemente o predicado *bridges(-ListOfBridges)* que devolve todos os voos que constituem uma ponte no grafo de voos.

m) Implemente o predicado *central_betweeness(-Nodes)* que devolve em *Nodes* uma lista com o nó ou nós que apresentam maior valor de *betweeness*. Este valor é medido como a quantidade de vezes que o nó faz parte do caminho mais curto entre dois outros nós.

n) Implemente o predicado *add_times(+Time, +Duration, -FinalTime, -NextDay)* que tem como argumentos uma hora inicial (no formato HHMM) e uma duração (em minutos), e devolve em *FinalTime* a hora final resultado de adicionar *Duration* a *Time*, e em *NextDay* uma *flag* (valor 0 ou 1) indicando se transitou para o dia seguinte.

o) Implemente o predicado *time_diff(+Time1, +Time2, -Duration)*, que devolve em *Duration* o tempo entre *Time1* e *Time2*. Note que se pode haver mudança de dia entre *Time1* e *Time2*.

p) Implemente o predicado *waiting_time(+Flight1, +Flight2, -WaitingTime)* que sucede se *Flight2* tem como origem o destino de *Flight1*, devolvendo em *WaitingTime* o tempo de espera, em minutos, entre os dois voos.

q) Implemente o predicado *filter_flights(+Origin, +Destination, +MaxWait, +MaxStops, -Fs)* que devolve em *Fs* as opções de voos entre *Origin* e *Destination*, tendo no máximo *MaxStops* escalas, e sendo o tempo máximo de espera em cada escala de *MaxWait* minutos.

r) Implemente o predicado *get_ordered_flights(+Origin, +Destination, -Flights)* que devolve em *Flights* as opções de voos entre *Origin* e *Destination* ordenadas ascendentemente pelo tempo total de voo.

3. Missionários e Canibais

Três missionários e três canibais pretendem atravessar da margem esquerda do rio para a margem direita, tendo apenas disponível um barco com capacidade para 2 pessoas para fazer a travessia. Implemente o predicado *missionaries_and_cannibals(-Moves)* que determine os movimentos a realizar para conseguirem fazer a travessia, sabendo que o número de canibais não pode ser superior ao número de missionários em qualquer das margens do rio.

4. Puzzle Deslizante

Implemente o predicado *sliding_puzzle(+Initial, -Moves)* que recebe como argumento a disposição inicial de um puzzle deslizante (representado como uma lista de listas de tamanho *N*x*N*, em que as peças são representadas por um número de 1 a N^2-1, e a célula vazia por 0), e devolve uma lista com uma possível sequência de movimentos que permita resolver o puzzle. Os movimentos possíveis são deslizar para cima, baixo, esquerda ou direita.

5. Jogo do Nim

O jogo do Nim é jogado por dois jogadores, existindo inicialmente um número arbitrário de pilhas, cada uma delas com um número arbitrário de fósforos. Em cada jogada, um jogador retira um ou mais fósforos de uma das pilhas existentes. O vencedor é o jogador que retira os últimos fósforos.

Implemente o predicado *winning_move(+State, -Move)*, que recebe um estado (lista de inteiros, indicando o número de fósforos em cada pilha) e retorna uma jogada (número de fósforos a retirar e pilha da qual os retirar), se existir, que permita vencer sempre o jogo (quaisquer que sejam as jogadas do adversário nas jogadas seguintes).

6. Subir escadas

O Asdrúbal sobe as escadas de entrada para sua casa sempre de uma forma diferente, um ou dois degraus de cada passada. Implemente o predicado *steps(+Steps, -N, -L)* que recebe o número de degraus da escada, e devolve em *N* o número de formas de as subir, e em *L* a lista com todas as possibilidades (cada possibilidade será uma lista com sequências de 1s e 2s).