Universidade Federal de Pernambuco

Curso de Graduação de Sistemas de Informação

Lógica para computação

Alunos:

Josef Jaeger<jjb>

Caio Elias<cerp>

Pedro Souza<pams>

Matheus Fillipe<mfsm>

Projeto de análise e elaboração em prolog

Relatório de detalhamento de funcionamento do código feito

Recife, Pernambuco

2022

Descrição do problema

Game of Thrones foi uma das séries de maior sucesso dos últimos anos e devido sua grande popularidade foi estabelecido para nós, alunos de lógica para computação, um projeto com o objetivo de realizar tarefas com fatos da série como por exemplo a qual casa um personagem pertence, para isso foi fornecido uma base de dados (anexada no fim do arquivo) contendo todos os personagens da série e os fatos relacionados a eles.

Para realizar essas atividades foi utilizada a linguagem de programação (prolog), uma linguagem declarativa, ou seja, ao invés de o programa estipular a maneira de chegar à solução passo-a-passo, como acontece nas linguagens procedimentais ou orientadas a objeto, ele fornece uma descrição do problema que se pretende computar utilizando uma coleção de fatos e regras (lógica) que indicam como deve ser resolvido o problema proposto.

Tarefas Propostas:

1 - Explicar o predicado tell_me_about(X).

2 - Criar um predicado house_of(X, Y) que retorna a qual casa o personagem é.

3 - Criar um predicado power_of(X, Y) que retorna o poder do clã.

4 - Criar um predicado is_single(X) que retorna se o personagem é solteiro ou não.

5 - Criar um predicado marriage_power(X, Y, Z) que retorna em Z quantos aliados o clã da pessoa passada em X obteria ao se casar com Y.

6 - Criar um predicado power_by_marriage(X, List, Power) que dado um clã X retorna uma lista de casamentos recomendados e o poder aumentado caso a realização dos casamentos.

Tarefa Extra:

Essa tarefa não é obrigatória, mas foi feita pelo grupo.

Foi criado um QUIZ de fatos sobre Game of Thrones que pode ser iniciado no programa.pl digitando → quiz.

Tarefa 1

Sobre o predicado tell_me_about :

O que faz?

O predicado tell_me_about(X) recebe como parâmetro um personagem da série e devolve para o usuário se ele está vivo, se tem pai, mãe, irmãos e filhos.

Predicados Auxiliares:

alive_or_dead(X), parents(X,Y), children(X,Y), list_siblings(X,Y).

Devido às várias informações retornadas pelo predicado tell_me_about, ele conta com os predicados auxiliares listados acima que serão explicados posteriormente abaixo.

Funcionamento:

O predicado **"alive_or_dead"** consulta na base fatos o predicado **"status"** sobre esse personagem e retorna alive caso vivo e dead caso morto.

```
alive_or_dead(X) :-
      status(X, Y),
      format("Status: ~w", [Y]), nl.
```

Para retornar o pai e mãe do personagem, o predicado auxiliar **"parents"** é utilizado de forma que o nome do indivíduo é passado do lado direito do predicado → **parent(**Pais, nome**)** que retorna assim seus pais. Por fim, é utilizado o método setof em parent para colocá-los em uma lista.

```
parents(X, Parents) :-
      setof(Y, parent(Y, X), Parents),
      !.

parents(X, Parents) :-
      not(setof(Y, parent(Y, X), Parents)),
      Parents = unknown.
```

Para retornar os filhos do personagem, tell_me_about utiliza o predicado **"children",** que também usa o **"parent"** passando o personagem do lado esquerdo, ou seja, onde ele é pai. Dessa forma, é retornado todos os seus filhos em uma lista utilizando o método setof.

```
children(X, Children) :-
      setof(Y, parent(X,Y), Children),
      !.

children(X, Children) :-
      not(setof(Y, parent(X,Y), Children)),
      Children = none.
```

Para realizar a tarefa de retornar os irmãos do personagem é utilizado o predicado **list_siblings** que utiliza o predicado **sibling** que por sua vez utiliza dois predicados **parent** e um predicado **dif.**

```
list_siblings(X, Siblings) :-
      setof(Y, sibling(X,Y), Siblings);
      Siblings = none.
```

```
sibling(X, Y) :-
      parent(Z, X),
      parent(Z, Y),
      dif(X, Y).
```

passo-a-passo do funcionamento de list_siblings:

1 - Para checar se personagens são irmãos primeiro o personagem é passado do lado direito do primeiro predicado **parent**, o valor retornado desse predicado são os pais do personagem.

2 - Esses pais são passados do lado esquerdo do outro predicado **parent**, que retorna os filhos desses pais.

3 - Por fim, o predicado **dif** impede que o nome do próprio personagem seja retornado já que ele também é filho desses pais.

4 - Dessa forma, os filhos desses pais são retornados e por consequência, são irmãos do personagem passado como parâmetro. O predicado **list_siblings** também retorna uma lista de irmãos e conta com o método setof no predicado sibling para realizar essa tarefa.

Print da utilização dos predicados auxiliares do tell_me_about.

```
tell_me_about(X) :-
      alive_or_dead(X),
      parents(X, Parents),
      format("Parents: ~w", [Parents]), nl,
      children(X, Children),
      format("Children: ~w", [Children]), nl,
      list_siblings(X, Siblings),
      format("Siblings: ~w", [Siblings]), nl,
      !.
```

Tarefa 2

Criação do predicado house_of:

O predicado **“house_of**“ recebe um nome e utiliza o sobrenome do personagem para indicar a qual clã ele pertence, para fazer a quebra do nome foi utilizado o método **atomic_list_concat.**

Observação: Como em alguns clãs existem pessoas com sobrenomes diferentes foi tratado cada exceção para seguir fielmente as casas como descrito dentro do arquivo.

Então, o predicado ficou da seguinte forma:

```
house_of(Z, W):-
      atomic_list_concat(L, '_', Z), nth1(2, L, S),
      ( S==snow, (W=stark; W=targaryen);
        S==sand, W=martell;
        S==of, W=martell;
        S==harlaw, W=greyjoy;
        S==estermont, W=baratheon;
        S==hightower, W=tyrell;
        S==lannister, W=S;
        S==baratheon, W=S;
        S==greyjoy, W=S;
        S==stark, W=S;
        S==martell, W=S;
        S==targaryen, W=S;
        S==tyrell, W=S;
        Z==aegon_targaryen, W=martell;
        Z==rhaenys_targaryen, W=martell); Z==gendry, W=baratheon.
```

Tarefa 3

Criação do predicado power_of:

O predicado house_of(X,Y) recebe em X um clã e devolve o poder deste clã em Y.

Para o funcionamento do predicado foram necessários 4 predicados auxiliares: **split**, **tamanho, membros_cla** e **status.**

O funcionamento de cada um se dá da seguinte forma:

1 - O predicado **“membros_cla”** usa o predicado **“status”** para retornar o nome de todas as pessoas da base de dados.

2 - Em seguida dentro do predicado **“split”** esses nomes são passados e separados entre nome e sobrenome. Dessa forma, os sobrenomes indicam a qual clã a pessoa pertence. Com esses fatos em mãos, basta fazer com que os nomes que têm o sobrenome do clã passado como parâmetro sejam retornados fazendo uma comparação.

3 - Dentro do predicado **“power_of”** é utilizado o método setof no predicado **“membros_cla”** que gera uma lista de todas as pessoas do clã.

4 - Por fim, o predicado **“tamanho”** conta o número de pessoas na lista gerada no setof para retornar o poder do clã.

```
split(X, Z) :- (atomic_list_concat(L, '_', X), nth1(2, L, S)), Z=S.

tamanho([],0).
tamanho([_|W],N):- tamanho(W,N1), N is N1+1.

membros_cla_tyrell(X, Y) :- (status(A, _), split(A, Z)),  (Z==X; Z==hightower), Y=A.
membros_cla_stark(X, Y) :- (status(A, _), split(A, Z)),  (Z==X; Z==snow), Y=A.
membros_cla_martell(X, Y) :- (status(A, _), split(A, Z)),  (Z==X; Z==sand; Z==of;A==aegon_targaryen; A==rhaenys_targaryen), Y=A.
membros_cla_baratheon(X, Y) :- (status(A, _), split(A, Z)),  (Z==X; Z==estermont), Y=A.
membros_cla_lannister(X, Y) :- (status(A, _), split(A, Z)),  (Z==X), Y=A.
membros_cla_targaryen(X, Y) :- (status(A, _), split(A, Z)),  (Z==X; Z=='V'; Z==snow), Y=A.
membros_cla_greyjoy(X, Y) :- (status(A, _), split(A, Z)),  (Z==X; Z==harlaw), Y=A.

power_of(tyrell, Z) :- (setof(Y, membros_cla_tyrell(tyrell, Y), W)), tamanho(W, N), Z=N.
power_of(stark, Z) :- (setof(Y, membros_cla_stark(stark, Y), W)), tamanho(W, N), Z=N.
power_of(greyjoy, Z) :- (setof(Y, membros_cla_greyjoy(greyjoy, Y), W)), tamanho(W, N), Z=N.
power_of(baratheon, Z) :- (setof(Y, membros_cla_baratheon(baratheon, Y), W)), tamanho(W, N), Z is N + 1.
power_of(martell, Z) :- (setof(Y, membros_cla_martell(martell, Y), W)), tamanho(W, N), Z=N.
power_of(lannister, Z) :- (setof(Y, membros_cla_lannister(lannister, Y), W)), tamanho(W, N), Z=N.
power_of(targaryen, Z) :- (setof(Y, membros_cla_targaryen(targaryen, Y), W)), tamanho(W, N), Z=N.
```

Observação:

Note por exemplo no clã martell que algumas exceções tem que ser tratadas porque existem pessoas com sobrenome sand, norvos e targaryen que também estão na casa.

Tarefa 4

Criação do predicado is_single:

O predicado is_single(X) recebe como parâmetro uma pessoa e retorna se ela é solteira ou não.

Para o funcionamento do predicado **“is_single”** é utilizado o predicado **“parent”** passando o indivíduo do lado esquerdo e um underscore do lado direito. Dessa forma, caso o indivíduo tenha um filho será retornado true e false caso ela não tenha um filho, ou seja, ela é solteira.

foi utilizado o **“is__single”** que retorna true e um print “casado(a)” quando se tem filhos. Com essa ideia em mente esse predicado foi utilizado dentro do is_sinlge original da seguinte maneira → not(is__single) que retorna false caso seja casado.

```
is__single(X) :- parent(X,_),
      format(" "), nl,
      format(" casado(a)!"), nl,
      format(" "), nl.

is_single(X) :- not(is__single(X)),
      format(" "), nl,
      format(" solteiro(a)!"), nl,
      format(" "), nl.
```

Tarefa 5

Criação do predicado marriage_power(X,Y,Z):

Retorna em Z quantos aliados o clã da pessoa passada em X obteria ao se casar com Y.

Para realizar a tarefa 5 foi necessário tratar as exceções pedidas no projeto.

exceções: (Não pode casar os gêneros iguais, não pode se casar gente do mesmo clã e não pode ter gente já casada).

Predicados utilizados para tratar as exceções:

1 - **teste_cla(X, Y)** --> usa o predicado **house_of** e compara o sobrenome pra ver se são do mesmo clã. (Print na tarefa 2 do house of)

```
verifica_cla(X, Y) :- (house_of(X,S), house_of(Y, V)), S == V,
      format("    "), nl,
      format("O casamento entre membros do mesmo cla nao e permitido !!"), nl,
      format("    "), nl.

teste_cla(X, Y) :- not(verifica_cla(X, Y)).
```

2 - **teste_casamento(X, Y)** --> utiliza os predicados male e female, e retorna true quando são de gêneros diferentes.

```
casamento(X, Y) :- ((male(X), male(Y)) ; (female(X), female(Y))),
      format("    "), nl,
      format("O casamento de mesmo genero nao e permitido neste reino !!"), nl,
      format("    "), nl.

teste_casamento(X, Y) :- not(casamento(X,Y)).
```

3 - **teste_solteiro(X)** --> checa se ambos personagens são solteiros e retorna true caso sim.

```
eh_solteiro(X) :- parent(X,_),
      format("                          "), nl,
      (write(X), write(" ja e casado !!")),
      format("                          "), nl,
      format("                           "), nl.

teste_solteiro(X) :- not(eh_solteiro(X)).
```

Os predicado utilizado para retornar os aliados:

1 - **retorna_lista_completa(X, T)** --> utiliza os predicados: **uncle, aunt, ancestor, neice, nephew e list_siblings.** Esse predicado retorna todos os parentescos solicitados: (irmãos, tio, tia, sobrinho, sobrinha, ancestrais).

2 - **list_soma(X, U)** --> usa o método setof no predicado **retorna_lista_completa** criando uma lista sem redundância com todos os parentes da pessoa passada como parâmetro. Após a criação da lista é utilizado o predicado tamanho para calcular quantos elementos tem na lista, devolvendo assim o poder aliado.

```
retorna_lista_completa(X, T):-
      ((aunt(U, X), T=U);
      (uncle(Y, X), T=Y);
      (ancestor(H, X), T=H);
      (sibling(K, X), T=K);
      (neice(J, X), T=J);
      (nephew(L, X), T=L)).

lista_soma(X, U) :- setof(G, retorna_lista_completa(X, G), W), tamanho(W,N), U=N.
```

Marriage_power:

marriage_power(nome_1, nome_2, Poder) --> utiliza todos os predicados de exceções e o **list_soma(X,U).** Dessa forma, se nome_1 for solteiro, nome_2 solteiro, ambos de gênero diferente e de clãs distintos o **list_soma(X, U)** será ativado passando o nome_2 como parâmetro e devolverá como resposta os aliados que ele trará incluindo ele.

```
marriage_power(X, Y, Z) :-
      (teste_casamento(X, Y),
      teste_solteiro(X),
      teste_solteiro(Y),
      teste_cla(X, Y)),
      lista_soma(Y, U),
      Z is U+1.
```

Tarefa 6

Criação do predicado power_by_marriage(X, Y, Z):

O predicado **power_by_marriage** recebe em X um clã e retorna uma lista de casamentos recomendados para aumentar o poder desse clã em Y. Em Z é retornado o poder que seria recebido caso realizado os casamentos da lista.

Para efetuar essa tarefa não conseguimos uma solução automatizada que a realizasse em Prolog pela falta de costume com a linguagem. Porém, utilizamos predicados para nos retornar os solteiros de cada clã. O **marriage_power** foi utilizado para retornar quantos aliados o indivíduo retorna e dessa forma montamos uma lista de forma manual com os fatos em mãos.

Para nos retornar os solteiros de cada clã:

```
solteiro(X) :- not(parent(X,_)).

%____Retorna os solteiros de cada clã____

solteiros_tyrell(X, Y) :- (status(A, _), split(A, Z)),  ((Z==X; Z==hightower), solteiro(A)), Y=A.

solteiros_stark(X, Y) :- (status(A, _), split(A, Z)),  ((Z==X; Z==snow), solteiro(A)), Y=A.

solteiros_lannister(X, Y) :- (status(A, _), split(A, Z)),  ((Z==X), solteiro(A)), Y=A.

solteiros_martell(X, Y) :- (status(A, _), split(A, Z)),  ((Z==X; Z==sand; Z==of), solteiro(A)), Y=A.

solteiros_baratheon(X, Y) :- (status(A, _), split(A, Z)),  ((Z==X; Z==estermont), solteiro(A)), Y=A.

solteiros_greyjoy(X, Y) :- (status(A, _), split(A, Z)),  ((Z==X), solteiro(A)), Y=A.

solteiros_targaryen(X, Y) :- (status(A, _), split(A, Z)),  ((Z==X; Z=='V'; Z==snow), solteiro(A)), Y=A.
```

```
power_by_marriage(Cla, Lista, Poder) :-
      casamento_recomendado(Cla, List, Power),
      format(" "), nl,
      format("Casamentos recomendados e aliados ganhos:"), nl,
      format(" "), nl,
      Lista=List,Poder=Power.
```

Tarefa Extra

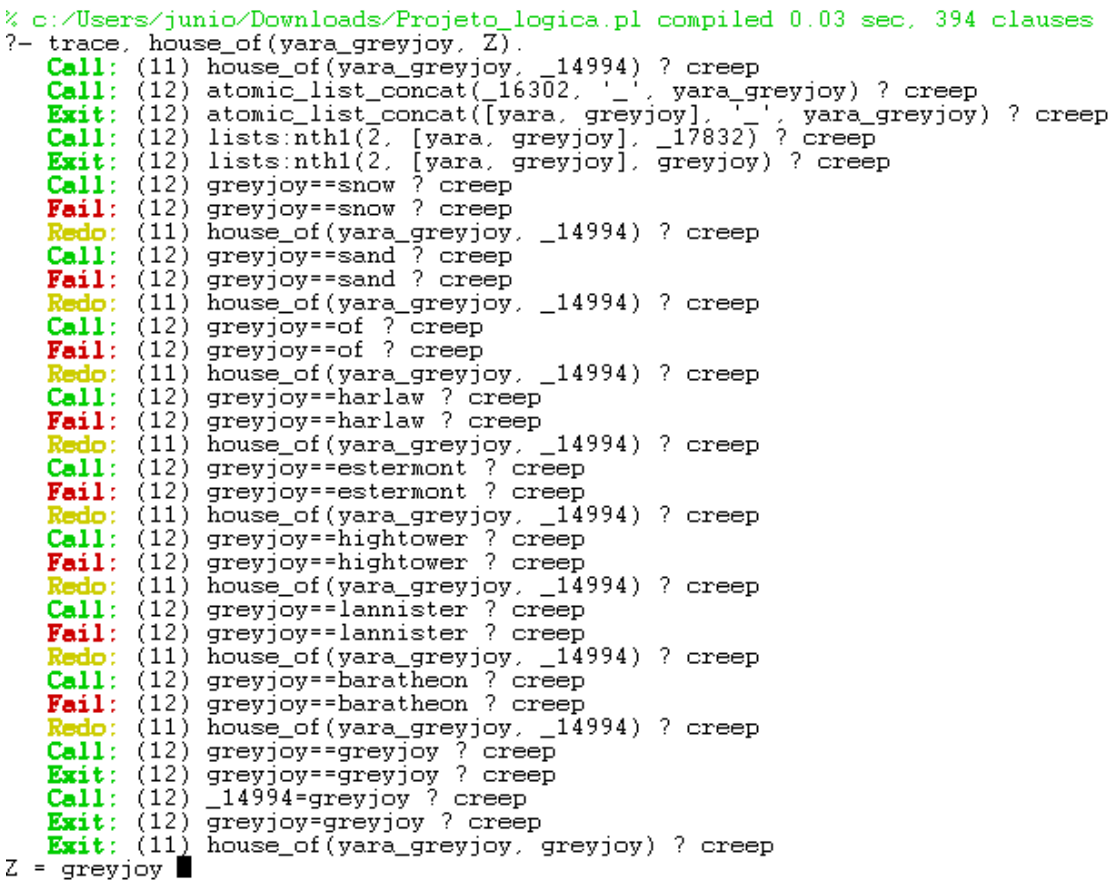
Foi criado um QUIZ de GoT utilizando comandos read para ler entradas e write para printar as respostas. Foi definido também alguns predicados para definir as respostas corretas.

O comando para iniciá-lo é → quiz.

```
% c:/Users/junio/Downloads/Projeto_logica.pl compiled 0.02 sec, 390 clauses
?- quiz.

INSTRUCAO: Coloque ponto ao final de cada resposta para funcionar!!

                                GoT QUIZ

                Vamos ver se voce e mesmo fa de Game of Thrones!

Preparado(a)?
|: sim.
                            VAMOS NESSA !

                1 - Quem pediu para Melisandre ressucitar Jon snow?

                a) Tormund
                b) Edd Doloroso
                c) Sor Davos
                d) Meitsre Aemon
                e) Sam Tarly

|: 
```

Execução do **house_of** com **Trace** ligado.

Note no print que o programa separa o sobrenome do nome e vai realizando todas as comparações estabelecidas até que pelo menos uma seja verdadeira.



Não será mostrado os outros Traces porque alguns dos predicados retornam muitas linhas de debug e isso deixaria o relatório extenso. Por esse motivo foi selecionado o house of que não é o menor predicado nem o maior.

Bibliografias utilizadas:

Base do projeto: https://github.com/rachelwiles/GoT-Check

Documentação da linguagem: https://www.swi-prolog.org/

Youtube:https://www.youtube.com/watch?v=SzQl4syxWtg&list=PLNQv4RjMoSKg_4IV1NxTR28K_c0yC_PHD