

**Trabalho 1 – Programação em Lógica**

Bruno Pereira, 69303

João Mano, 69854

Patrícia Rocha, 69636

**Resumo**

Este relatório tem como objetivo descrever e explicar todo o desenvolvimento das tarefas requeridas no enunciado do primeiro exercício prático da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio.

Referem-se todas as decisões tomadas durante a realização deste exercício da forma mais detalhada possível.

Aplicam-se conhecimentos leccionados nas aulas práticas e teóricas, de maneira a resolver todos as tarefas propostas de uma forma correta.

Tabela de Conteúdos

**Trabalho 1 – Programação em Lógica5**

Introdução**5**

Objetivos**5**

Preliminares**5**

Descrição do Trabalho e Análise de Resultados**6**

Base de Conhecimento**6**

Implementação dos Predicados**7**

Inserção de Conhecimento**7**

Remoção de Conhecimento**8**

Relações Familiares**8**

Invariantes**9**

Exemplo Práticos e Análise de Resultados**9**

Conclusões e Sugestões**13**

Tabela de Figuras

**Exemplos Práticos e Análise de Resultados10**

**Figura 1 –** Inserção de novo conhecimento**11**

**Figura 2 –** Inserção de conhecimento repetido**11**

**Figura 3 –** Inserção de 3 pais**11**

**Figura 4 –** Determinar grau de relação**11**

**Figura 5 –** Determinar relação**11**

**Figura 6 –** Testar se duas pessoas possuem a mesma naturalidade**12**

**Figura 7 –** Remoção de um facto**12**

**Trabalho 1 – Programação em Lógica**

Este documento visa a explicação e documentação do processo de resolução dos exercícios propostos na unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio de forma clara e completa.

Serão expostos todos os parâmetros considerados importantes e relevantes à resolução dos exercícios.

1. **Introdução**

A Programação em Lógica é um tipo especfico de programação cujo objetivo é a concetualização de um programa cujo conteúdo se prende em factos (registos que se sabem verdadeiros), predicados (associados aos factos) e regras. A este programa podem ser estruturadas questões sobre o seu conteúdo e obter-se-ão respostas válidas e corroboradas pela lógica em si.

A programação em lógica baseia-se em dois princípios básicos para a “descoberta” das respostas (soluções) a essas questões: Lógica, usada para representar os conhecimentos e informação, e Inferência, regras aplicadas à Lógica para manipular o conhecimento.

1. **Objetivos**

Com o objetivo de consolidar conhecimentos e obter experiência e prática face a problemas de programação em lógica foram realizadas todas as tarefas pertencentes ao primeiro exercício prático da UC SRCR.

O intuito deste exercício prende-se na construção de um programa capaz de armazenar conhecimento sobre uma árvore genealógica e através deste solucionar questões deste tema.

1. **Preliminares**

A presença nas aulas teórica e práticas desta unidade curricular foi um fator crucial no sucesso da realização deste trabalho, adicionando a isto existe uma breve pesquisa sobre Programação em Lógica e exemplos do seu uso, por forma a captar e entender o que deveria ser formado, esta pesquisa incidiu de uma forma quase global nos apontamentos das aulas teóricas e práticas.

De modo a que a leitura deste documento seja perceptível em termos de conceitos e símbolos é necessário fazer referências breves a noções básicas de ProLog, a linguagem em que é desenvolvido este trabalho.

Tal como foi mencionado anteriormente uma linguagem de programação lógica utiliza a lógica para representar conhecimento e inferências para manipular informação. Um programa neste tipo de programação possui então os seguintes parâmetros:

* Factos – constatações sobre algo que se conhece e se sabe verdadeiro, por exemplo cor( azul );
* Predicados – implementam relações, por exemplo o predicado filho( filho, pai ) implementa a relação de descendência direta (ser filho de);
* Regras – utilizadas para definir novos predicados.

Quanto a este último parâmetro, regras, é necessário entender o significado de cada um dos símbolos utilizados. Tem-se então que:

* **.** utilizado para terminar uma declaração;
* **:-** significa “se”;
* **,** possui o significado “e”;
* **;** significa “ou”;
* **//** representa a unificação;

É ainda necessário referir que as variáveis representam-se por maiúsculas e constantes, predicados e factos com minúsculas.

Com estas noções como base passa-se agora ao desenvolvimento das tarefas do exercício.

1. **Descrição do Trabalho e Análise de Resultados**

Nesta parte do documento serão explicitadas todas as etapas de resolução dos desafios fornecidos bem como todas as decisões efetuadas no processo.

* 1. **Base de Conhecimento**

Uma vez que tem-se como objetivo desenvolver um sistema de representação de conhecimento sobre uma árvore genealógica torna-se importante que sejam defenidos os seguintes elementos de conhecimento:

* Relação de progenitor (pai);
* Relação de avô ou avó (avo);
* Relação de irmão ou irmã (irmao);
* Relação de tio ou tia (tio);
* Relação de sobrinho ou sobrinha (sobrinho);
* Relação de bisavô ou bisavó (bisavo);
* Relação de trisavô ou trisavó (trisavo);
* Relação de tetravô ou tetravó (tetravo);
* Relação de neto (neto);
* Relação de descendente (descendente);
* Determinação do grau de descendência (grau);
* Relação de primo ou prima (primo);
* Determinação de naturalidade (naturalidade);
* Determinação mesma naturalidade (equalNaturalidade);
* Determinação da relação entre duas pessoas (relacao);

Desta forma consegue-se representar toda a informação pedida nas subtarefas do exercício proposto. A implementação destes elementos serão, de seguida, apresentadas bem como exemplos de usos práticos.

* 1. **Implementação dos Predicados**
     1. **Inserção de Conhecimento**

Por forma a conseguir inserir conhecimento foi criado o predicado *evolucao* que utiliza os predicados *insere*, *teste* e *solucoes* e é implementado da seguinte forma:

*evolucao( T ) :-*

*solucoes(I,+T::I,S),*

*insere(T),*

*teste(S).*

Dos predicados que utiliza é de notar o do insere:

*insere(T):-*

*assert(T).*

*insere(T):- retract(T),!,fail.*

Ou seja, dado um facto T a inserir, são calculadas as soluções que recolhem todos os invariantes e, de seguida inserido o facto, é depois realizado o teste dos invariantes e caso falhem, dada a política de retract do ProLog, vai ser novamente utilizado o insere que, na segunda vez vai utilizar a segunda definição numa tentativa de encontrar o valor de verdadeiro e com isso retirar o facto anteriormente inserido. Evitando que (dado a política de retract) se entre num ciclo sem fim é utilizado o cut (!) para que após a primeira “entrada” nesta definição não seja realizado o retract e termine a prova.

O predicado *teste*, é bastante simples de implementar pois, dada uma lista de invariantes apenas testa o primeiro e de seguida recursivammente faz o teste da cauda da lista, sendo que teste de lista vazia é verdadeiro. No predicado de *solucoes* foi usado o *findall* disponibilizado pelo ProLog, que dado um teorema a demonstrar procura qualquer instanciação desse predicado, caso não exista é dado insucesso (fail).

* + 1. **Remoção de Conhecimento**

Para a remoção de conhecimento foi implementado o predicado remover:

*remover( F ) :-*

*retract(F).*

Que recorre ao uso do predicado retract disponibilizado pelo ProLog, esta foi a decisão utilizada pois, dado que não se consideraram nenhuns invariantes de remoção, achou-se que remover um facto, neste contexto, deveria ser sempre possível.

* + 1. **Relações Familiares**

Neste subtópico apenas serão apresentadas as implementações que consideramos mais importantes. Tendo isto como base, iniciamos então com o predicado *pai*, implementado à custa do predicado *filho*, não será apresentado o predicado *filho* pois trata-se apenas da representação de um predicado facto, sem condições e, por isto, bastante simples.

*pai( P,F ) :-*

*filho( F,P ).*

A este predicado estão associados dois invariantes, o de conhecimento repetido e de limitação de pais (ver secção 4.2.4).

Foram ainda implementados predicados *avo*, *bisavo*, *trisavo*, *tetravo* mas, como a semelhança é grande apenas apresentaremos o *avo*:

*avo( A,N ) :-*

*pai( X, N ) , pai( A,X ).*

Um indivíduo A é avô ou avó de um N, se o A for pai de um indivíduo X e N for filho desse indivíduo. Tal como no pai estão associados os invariantes de conhecimento repetido e limitação de avós, isto é dado para os restantes predicados mas multiplicando por dois o número de avós permitidos, ou seja, quatro avós, oito bisavós, etc.

O predicado *irmao*, *tio* e *primo* possuem as seguintes implementações e são sujeitos apenas aos invariantes de repetição de conhecimento:

*irmao(X,Y):-*

*pai(P,X),*

*pai(P,Y),*

*X\==Y.*

*tio(T,S):-*

*irmao(T,X),*

*pai(X,S).*

*primo(P1,P2):-*

*pai(X,P1),*

*tio(X,P2).*

Considerou-se que um indivíduo não pode ser irmão de ele próprio e que só é irmão se tiver um mesmo progenitor, um tio de S, só é tio se for irmão de um individuo X que seja pai de S e, um primo só é primo de P2 se o seu pai for tio de P2.

As implementações de *sobrinho* e *neto* foram concetualizadas à custa de *tio* e *avo* respetivamente.

O predicao *relacao* é de uma simplicidade extrema e por isso não será apresentado, este é concetualizado pela exaustão das relações existentes, ou seja, relacao( X, Y, R ) só é verdadeiro se R( X, Y ), sendo que R seria percorrido pelas relações existentes. É então definido, tendo que existem n tipos de relações, através de n declarações.

Para os predicados descendente, ascendente e grau será apresentada a implementação de todos excepto ascendente, pois esta é à custa da implementação de descendente.

*descendente( D,A ) :-*

*pai(A,D).*

*descendente( D,A ) :-*

*pai( X,D ) , descendente( X,A ).*

*grau( D,A,1 ) :-*

*pai( A,D ).*

*grau( D,A,G ) :-*

*pai( D,X ) , grau( X,A,N ), N is G+1.*

Entenda-se que ambas as implementações têm 2 predicados, ou seja, no caso do descendente, este pode ser descendente diretamente (filho) e daqui surge o primeiro predicado utilizando o predicado *filho*,ou não e daqui surge o segundo predicado que dita que um indivíduo D é descendente de A se D for filho de um X de forma a que X seja descendente de A.

No caso do predicado grau é o mesmo algoritmo que o anterior mas por cada “subida” na árvore genealógica o grau é incrementado (N is G+1), o caso de paragem estabelece-se na descendência direta pois o grau de um filho para o seu pai é 1.

Finalmente apresenta-se o predicado *naturalidade* e *equalNaturalidade*. Tal como no *filho* o predicado *naturalidade* é construído através de predicado factos por exemplo: naturalidade( joao, braga ). Já o predicado *equalNaturalidade* baseia-se no *naturalidade* de forma a que só retorna verdadeiro se a naturalidade dos sujeitos for idêntica e se os sujeitos não forem o mesmo (por defeito achou-se que não se deveria considerar quando o teste fosse efetuado com o mesmo indivíduo).

*equalNaturalidade(P1,P2) :-*

*naturalidade(P1,X),*

*naturalidade(P2,X), P1\==P2.*

* + 1. **Invariantes**

Os invariantes concebidos para este trabalho foram apenas de inserção, apenas serão apresentados três pois os outros são bastante idênticos entre si. Existem então dois tipos de invariantes construídos os invariantes de repetição de conhecimento e os de limitação, os primeiros é para evitar que se introduza conhecimento repetido, o segundo é para corresponder à realidade, dado que um filho só tem dois pais, quatro avós, etc.

Um invariante de repetição de conhecimento:

*+avo( A,N ) :: (solucoes( ( A,N ), (avo( A,N )),S ),*

*comprimento( S,C ),*

*C==1*

*).*

Este invariante funciona da seguinte forma: procura todas as soluções que existem daquele tipo e indica que só pode haver uma.

Um invariante de limitação:

*+avo( A,N ) :: (solucoes( As, (avo( As,N )),S ),*

*comprimento( S,C ),*

*C=<4*

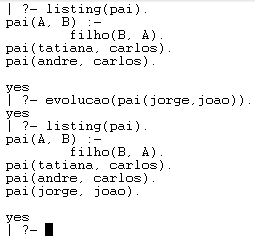
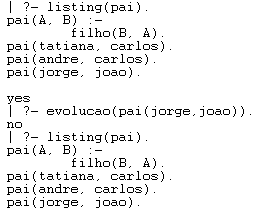
*).*

Este último funciona da mesma forma que o anterior mas procura não os predicados com aqueles termos, mas apenas com um dos termos (o que é limitado a um certo número de relação com outro) e diz que só podem existir n soluções (quatro no caso dos avós).

* + 1. **Exemplos Práticos e Análise de Resultados**

Nesta secção serão apresentados alguns exemplos práticos e discutido o resultado face ao esperado indicando a sua concordância ou discordância comparativamente ao esperado.

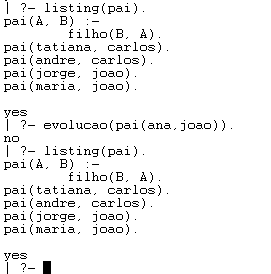
Inserção de um facto novo seguido da mesma inserção, neste caso é esperado que funcione a primeira tentativa e que na segunda seja retornado insucesso.

Os resultados são então os esperados.

**Figura 2 -** Inserção de conhecimento repetido

**Figura 1** - Inserção de novo conhecimento

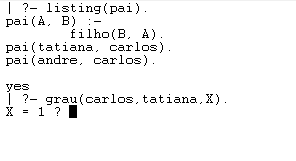
Um segundo exemplo prático será a inserção de 3 pais para um mesmo indivíduo que deverá retornar insucesso.



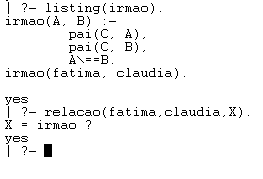
**Figura 3** - Inserção de 3 pais

Os resultados são também os esperados.

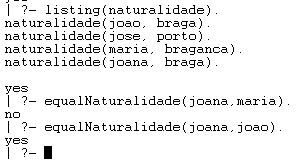
Os exemplos práticos finais são:

1. Determinar o grau da relação

**Figura 4** - Determinar grau de relação

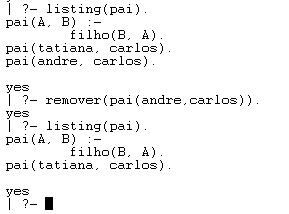
1. Determinar a relação

**Figura 5** - Determinar relação

1. Ver se dois indivíduos possuem a mesma naturalidade

**Figura 6** - Testar se duas pessoas possuem a mesma naturalidade

1. Remover um facto



**Figura 7** - Remoção de um facto

Os resultados são mais uma vez os esperados nestes últimos exemplos.

1. **Conclusões e Sugestões**

No final deste exercício prático consolidaram-se conceitos de programação em lógica através da prática neste tipo de programação. Em termos de resultados foram sempre os esperados, exceptuando-se uma situação que levou à aprendizagem do seguinte conceito: predicados usados para defenir novos predicados, influenciam em tudo a atuação do novo predicado. Tomando como exemplo o caso em que foi definido o predicado grau a partir do predicado filho e este não retornava o esperado para factos defenidos à custa do predicado pai. Isto acontece porque pai está definido à custa de filho mas filho não tem qualquer definição do predicado pai logo, quando usado o filho para defenir o grau, era possível obter os factos definidos por filho mas não os factos defenidos por pai. Alteramos assim todas as definições à custa de filho para pai, evitando futuros problemas.