

Escola de Engenharia

Sistemas de Representação de Conhecimento e Raciocínio

**TRABALHO PRÁTICO Nº 1**

Mestrado Integrado em Engenharia Informática

Grupo 30

**78416 Francisco José Moreira Oliveira**

**79617 Raul Vilas Boas**

**79175 Vitor Emanuel Carvalho Peixoto**

**Ano letivo 2017/2018**

Braga, Março de 2018

# Resumo

Este trabalho prático foi desenvolvido com o intuito de desenvolver e evoluir as competências adquiridas, na unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio, na programação em lógica com o PROLOG.

Este primeiro trabalho reflete sobre uma área de prestação de cuidados de saúde e o objetivo deste é desenvolver um sistema para este caso de estudo.

Este relatório foi desenvolvido com o intuito de explicar o processo de desenvolvimento, bem como as escolhas tomadas no decorrer deste.

Índice

[Resumo i](#_Toc509062209)

[1. Introdução 1](#_Toc509062210)

[2. Preliminares 2](#_Toc509062211)

[3. Descrição do trabalho e análise de resultados 3](#_Toc509062212)

[3.1 Representação do conhecimentos 3](#_Toc509062213)

[3.2 Registar utentes, prestadores e cuidados de Saúde 4](#_Toc509062214)

[3.3 Remover utentes, prestadores e cuidados de saúde 8](#_Toc509062215)

[3.4 Identificar utentes por critérios de seleção 9](#_Toc509062216)

[3.5 Identificar as instituições prestadores de cuidados de saúde 11](#_Toc509062217)

[3.6 Identificar cuidados de saúde prestados por Instituição/Cidade/Data. 12](#_Toc509062218)

[3.7 Identificar os Utentes de um Prestador/Especialidade/Instituição 13](#_Toc509062219)

[3.8 Identificar Cuidados de Saúde realizados por Utente/Prestador 14](#_Toc509062220)

[3.9 Determinar todas as Instituições/Prestadores a que um Utente já recorreu 15](#_Toc509062221)

[3.10 Calcular o Custo Total dos Cuidados de saúde por Utente/Especialidade/Prestador/Datas 16](#_Toc509062222)

[4. Extras 19](#_Toc509062223)

[5. Funções Auxiliares 22](#_Toc509062224)

[6. Conclusões e sugestões 23](#_Toc509062225)

[Referências 24](#_Toc509062226)

[Anexos 25](#_Toc509062227)

# Introdução

Este primeiro trabalho, consiste na realização de um conjunto de exercícios envolvendo um sistema desenvolvido à volta da área de prestação de cuidados de saúde e o objetivo deste é desenvolver um sistema de representação de conhecimento e raciocínio para este caso de estudo. Logo, para isso, vai ser necessário caracterizar o conhecimento como apresentado no enunciado.

Vai ser necessário representar utentes, prestadores e cuidados de saúde, onde cada um possui a sua própria informação. Para representar esse conhecimento foram definidos os predicadas para cada uma dessas entidades.

# Preliminares

Neste caso em específico, consideramos que não há a necessidade de explicar qualquer tipo de preliminares, uma vez que todo o conhecimento aplicado no desenvolvimento deste trabalho prático foi obtido unicamente através da matéria lecionada na unidade curricular.

No entanto, para a resolução de algumas funcionalidades extra, foi necessário a utilização de algumas funções pré-definidas no PROLOG, como por exemplo o “*sort* ”.

# Descrição do trabalho e análise de resultados

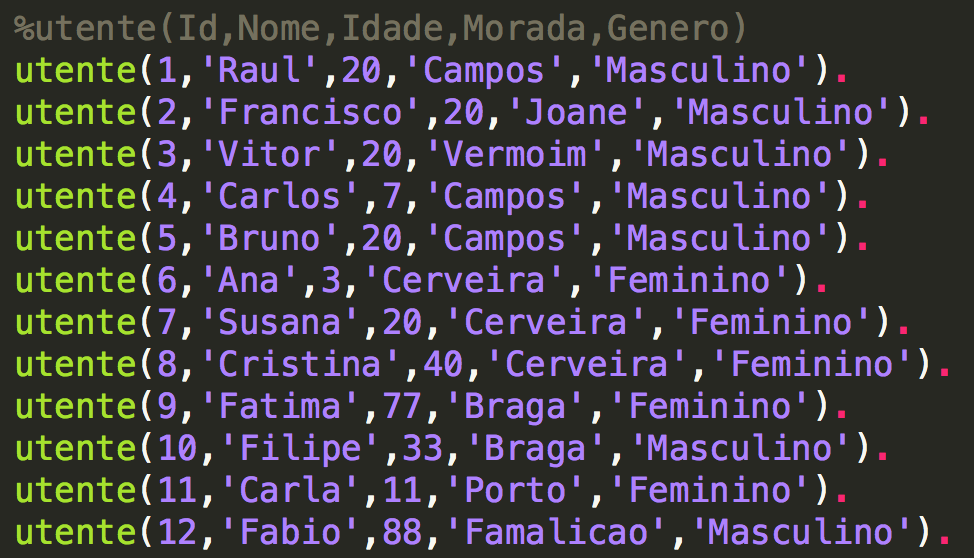
Neste trabalho existem 3 tipos de predicados que representam o conhecimento, sendo, eles os **utentes**, os **prestadores** e os **cuidados de saúde**. Os utentes possuem um identificador, um nome, uma idade e a morada. Os prestadores contêm um identificador, um nome, uma especialidade e uma instituição. E por fim, os cuidados de saúde possuem uma data, o identificador do utente, o identificar do prestador, uma descrição e um custo.

* utente: #IdUt, Nome, Idade, Morada ↝ { 𝕍, 𝔽 }
* prestador: #IdPrest, Nome, Especialidade, Instituição ↝ { 𝕍, 𝔽 }
* cuidado: Data, #IdUt, #IdPrest, Descrição, Custo ↝ { 𝕍, 𝔽 }

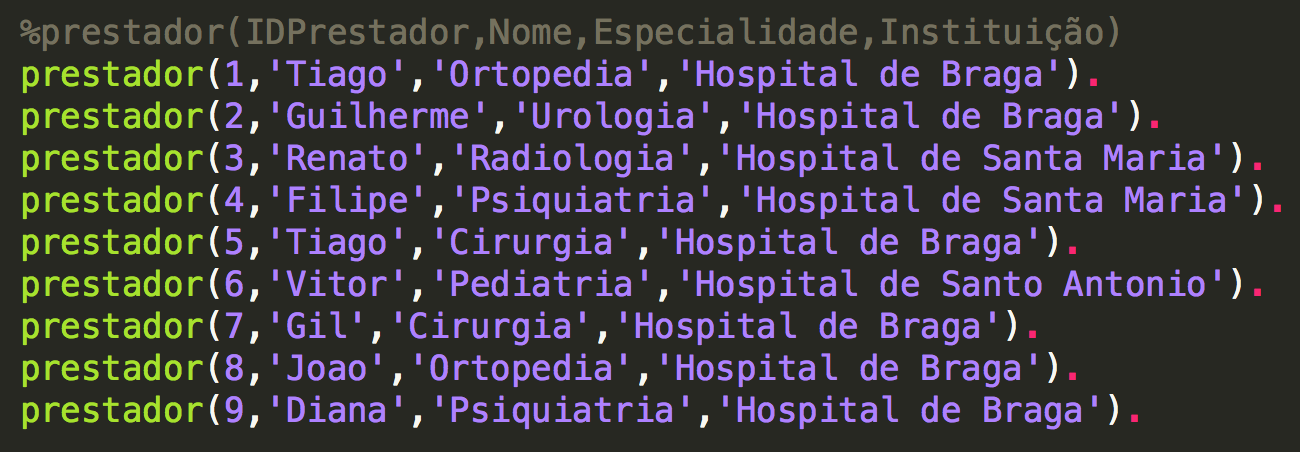
Para além disto, também foi adicionado nos utentes o tipo “género" que distingue os utentes do sexo masculino dos utentes do sexo feminino.

## Representação do conhecimento

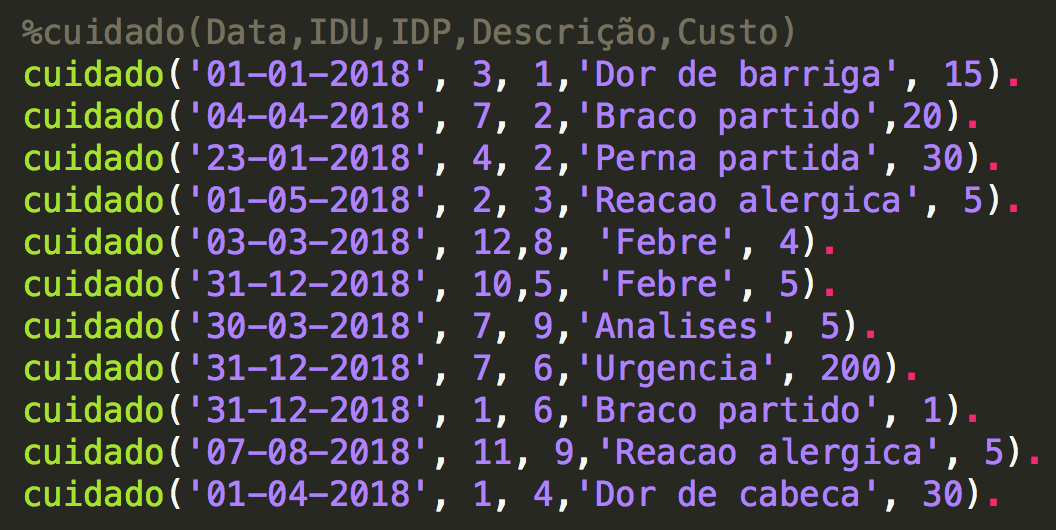
Para representar o conhecimento foi então necessário povoar a base de dados com exemplos de utentes, prestadores e cuidados de saúde.

 Primeiro começamos pelos utentes. Estes possuem identificador, nome, idade, morada e género:

**Figura 1** - Base de conhecimento dos utentes

Depois seguem-se os prestadores, caraterizados pelo identificador, nome, especialidade e instituição:

**Figura 2** - Base do conhecimento dos prestadores

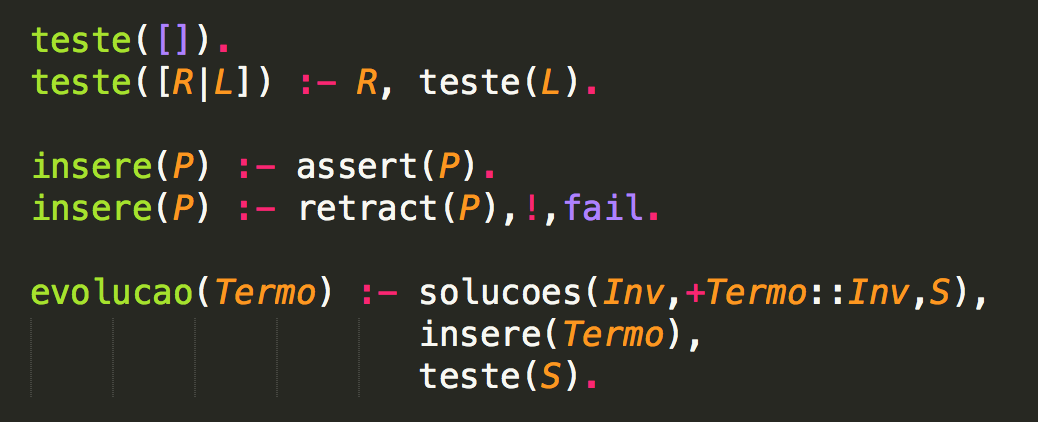
Por fim, temos os cuidados de saúde que possuem uma data, a identificação do utente e prestador, uma descrição e um custo:

**Figura 3** - Base do conhecimento dos cuidados de saúde

## Registar utentes, prestadores e cuidados de Saúde

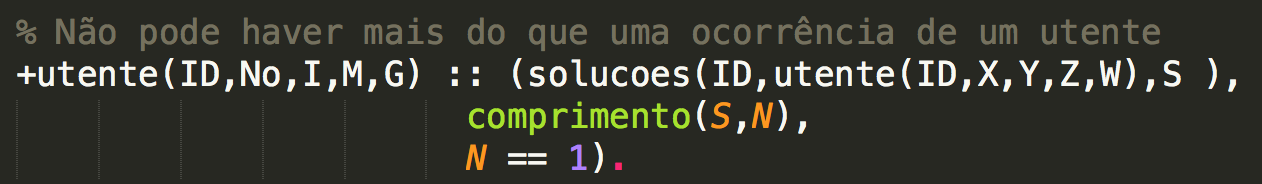
O primeiro ponto do enunciado pede para criar uma sintaxe que consiga registar utentes, prestadores e cuidados de saúde. Para além disso, esta inserção tem de respeitar certos aspetos, como por exemplo, não haver a possibilidade de inserir dois utentes com o mesmo identificador. Por esse motivo, também se criou invariantes que não permitam que isso aconteça.

Para isso utilizou-se a função evolução criada nas aulas práticas que permite a inserção de conhecimento na base de dados, com a ajuda da função “*assert*” do PROLOG, mas também permite ver se a inserção respeita todos os invariantes existentes. Caso o invariante não seja respeitado, o conhecimento, que foi anteriormente adicionado, é removido.

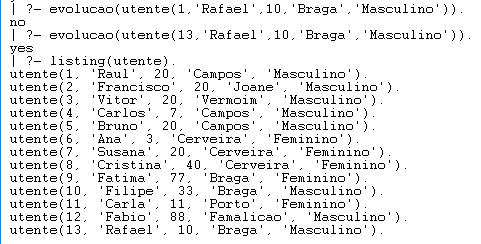


**Figura 4** - Código que permite a inserção de conhecimento

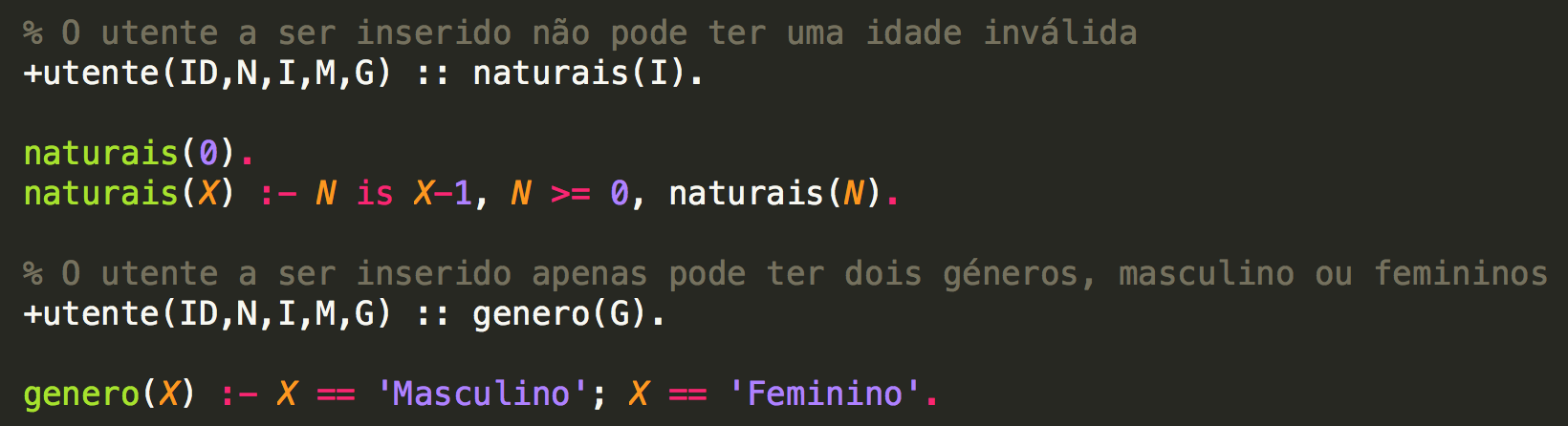
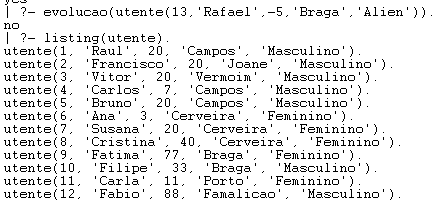
Logo, como não é possível que haja dois utentes com o mesmo identificador criou-se o seguinte algoritmo que dado um utente com um certo ID vai procurar na base de dados se há algum utente com esse mesmo ID, caso haja, coloca esse ID numa lista. Depois, é só analisar o tamanho da lista e verificar se a condição é respeitada.

Visto que, o conhecimento do utente em causa é adicionado antes dos invariantes serem verificados, o tamanho mínimo da lista será 1 que representará o utente que queremos adicionar. No entanto, se o tamanho for igual a 2, isto significa que já existia na base de conhecimento um utente com esse identificador e por esse motivo, não podemos adicionar o utente à base, sendo por isso removido.

**Figura 5** – Invariante que não permite inserção de conhecimento de utentes repetidos

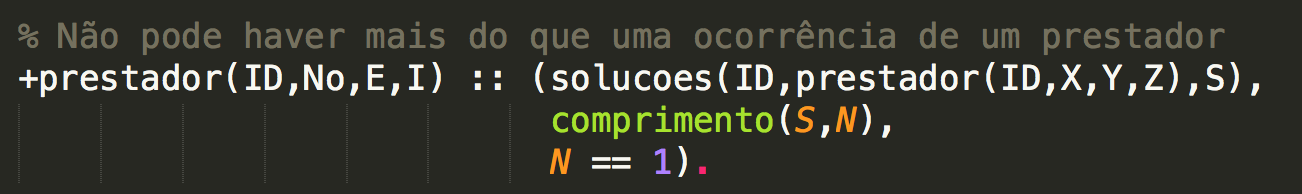
Para comprovar que o invariante estava correto, criou-se um exemplo em que se utilizou um identificador já usado na base de conhecimento (ID=1) e verificou-se que a sua inserção não acontecia. Depois, usando o mesmo exemplo, apenas alterando o identificador para um que não existia na base de conhecimento (ID = 13) verificou-se que o conhecimento foi inserido.

**Figura 6** - Resultado do invariante

Para além deste, também se criou outro invariante que não permite a inserção de um utente com uma idade inválida (que não pertença aos números naturais com o zero inclusive) e outro um que não permite que um utente seja de outro sexo para além de masculino ou feminino.

**Figura 8** - Resultado do invariante

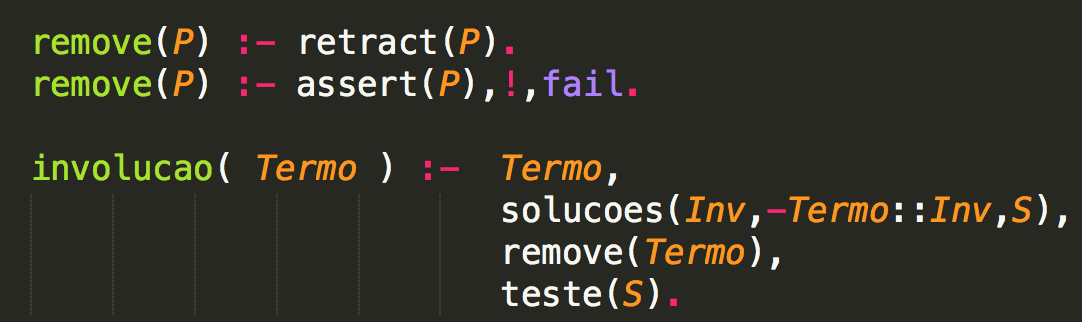
**Figura 7** - Invariante para a idade e para o género

Para o prestador, tal como para o utente, criou-se um invariante que não permite a inserção de um prestador com um identificador igual a um prestador já existente. A estrutura deste invariante é muito semelhante à do invariante anterior.

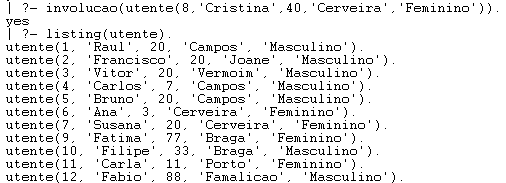
**Figura 9** - Invariante dos prestadores

Por último, para os cuidados criou-se três invariantes: O primeiro não permite a inserção de cuidados, caso o identificador do utente a ser inserido não exista; O segundo não permite a inserção caso o identificador do prestador não exista na base de conhecimento; O terceiro invariante não permite que o custo do cuidado a ser inserido seja inválido, isto é, menor que zero.

## Remover utentes, prestadores e cuidados de saúde

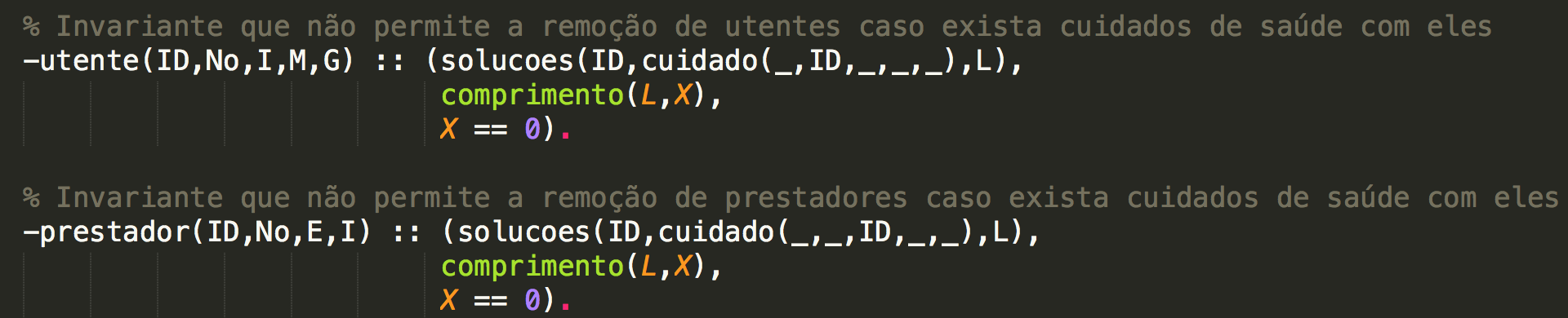
O seguinte ponto pede para remover utentes, prestadores e cuidados de saúde. Para isso criou-se então a seguinte função ‘involucao’ que permite que isso aconteça. Assim como a ‘evolucao’, a remoção do conhecimento também tem que se sujeitar a alguns invariantes que impedem que certas situações ocorram. Para além disto, também é importante referir que não é possível remover conhecimento que não exista na base de conhecimento. Para isso não acontecer, o argumento que a função ‘involucao’ recebe tem de ser verdadeiro, em que, apenas nesse caso o resto do predicado se irá realizar.

**Figura 11** - Código para a remoção de conhecimento

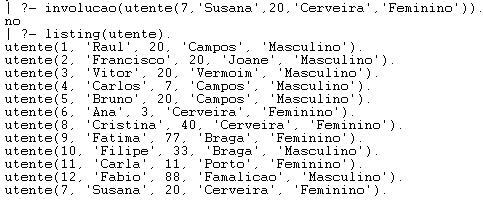


**Figura 12** - Resultado da remoção de conhecimento

Um dos casos em que se verificou que uma remoção seria inválida, era quando se tentaria remover um utente ou um prestador, quando existem cuidados de saúde com eles. Pois isso depois implicaria que haveria cuidados com utentes ou prestadores que não existiam na base de conhecimento e caso se quisesse saber a informações deles, seria impossível. Logo, criou-se dois invariantes que não permitem que se remova tanto um utente como um prestador, caso existam cuidados de saúdes com eles. Para isso, por exemplo, ao ser fornecido o utente para remover vamos procurar nos cuidados se há algum com o identificador do utente e caso haja é colocado numa lista. Depois, após calcular o comprimento da lista, retira-se as conclusões, isto é, caso o tamanho da lista seja maior ou igual a 1, quer dizer que existem cuidados com esse utente, caso o tamanho da lista seja 0, quer dizer que não existem cuidados com esse utente, e por isso, podemos removê-lo da base de conhecimento.



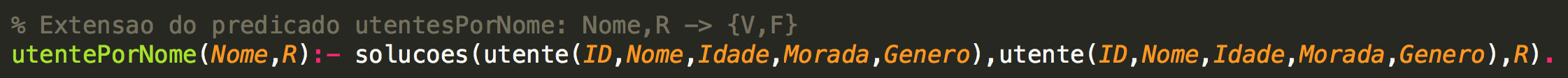
**Figura 13** - Invariantes de remoção

Visto que, como a Susana (utente com ID=7) está presente em dois cuidados de saúde não pode ser removida. No entanto, a sua posição no ‘listing’ é alterada porque o algoritmo primeiro remove e só depois de verificar se os invariantes foram todos respeitados é que volta a adicionar, caso isto não se verifique a inserção não é efetuada.

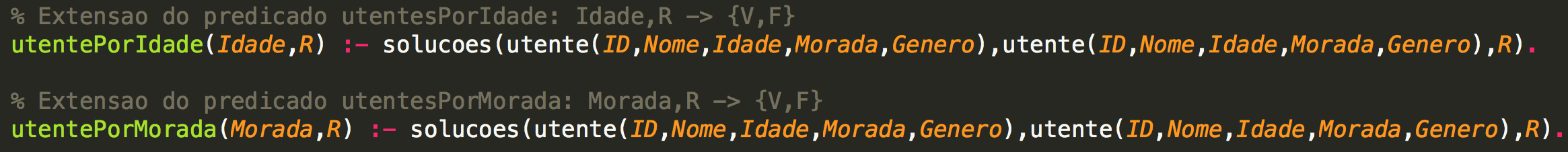
**Figura 14** - Exemplo dos invariantes

## Identificar utentes por critérios de seleção

Neste ponto, o exercício propunha a identificação dos diversos utentes pelos seus critérios de seleção como por exemplo, o nome, a idade, a morada e o género, sendo este último criado extra.

Para a seleção pelo nome criou-se o predicado ‘utentePorNome’ que recebe como argumento um nome e vai retornar uma lista com todas as informações de todos os utentes com esse nome. Para isso, a função vai comparar o nome recebido com o nome de todos os utentes e os utentes que tiverem o nome igual ao pretendido guarda-se numa lista que posteriormente é retornada como o resultado.

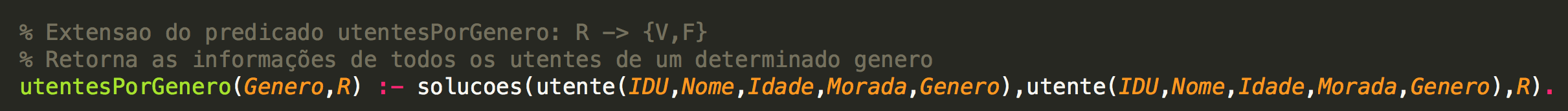
**Figura 15** – Função que seleciona por nome

Para o caso da idade e da morada, a estrutura do algoritmo é a mesma, apenas alterando o argumento recebido que será agora uma idade ou uma morada:

**Figura 16** - Funções que selecionam por idade e morada

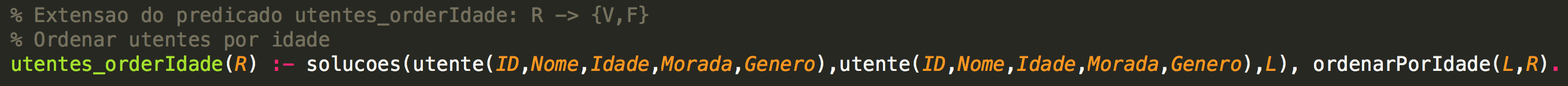
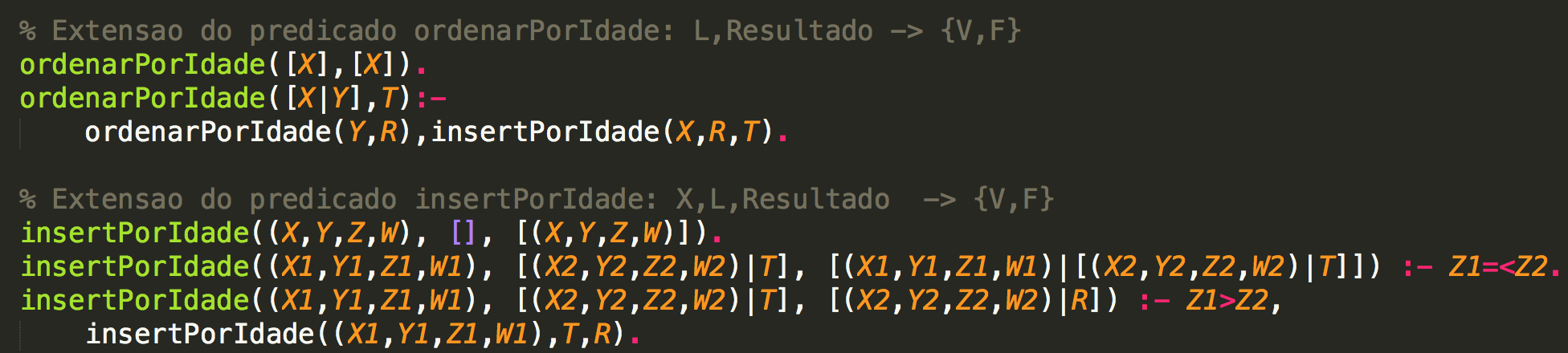
Podemos assim usar, por exemplo, a função ‘utentePorIdade’ e dar o argumento 20 e descobrir todos os utentes que têm vinte anos de idade:

**Figura 17** - Resultado obtido pela função ‘utentePorIdade’ para utentes com 20 anos.

Também se criou um predicado para o género, em que retorna todos os utentes de um determinado género:

**Figura 18** - Função que seleciona por género

Por último, criou-se um predicado que retornava as informações dos utentes ordenados crescentemente por idade, ‘utentes\_orderIdade’. Para isso utiliza-se uma função auxiliar, a ‘ordenarPorIdade’ que por usa vez utiliza outra função, denominada ‘insertPorIdade’. Com a combinação destas duas funções conseguimos com a lista de todas as informações de utentes ordena-los por idade, retornando assim o resultado pretendido.



**Figura 20** - Funções auxiliares que inserem utentes ordenados por idade.

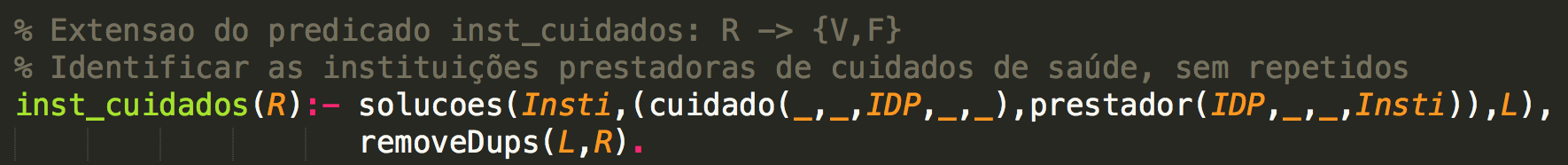
**Figura 19** - Função que retorna os utentes ordenados por idade

## Identificar as instituições prestadores de cuidados de saúde

**Figura 21** - Resultado da ‘utentes\_orderIdade’.

Neste exercício, é pedido para identificar as instituições que aparecem nos cuidados de saúde.

Como não é possível fazer uma ligação direta entre eles é necessário colocar mais argumentos no ‘solucoes’, isto é, ele vai primeiro descobrir todos os identificadores de prestadores que tem cuidados de saúde e retornar as instituições desses prestadores. No entanto, como ao fazer isto vai haver uma lista com as instituições repetidas usa-se a função ‘removeDups’ para remover as que estão repetidas.

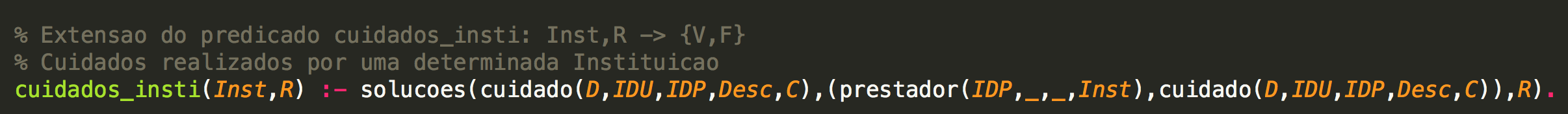
Obtendo assim, as instituições prestadoras de cuidados de saúde.

**Figura 23** - Resultado do predicado ‘inst\_cuidados’.

**Figura 22** - Identifica as instituições que prestam cuidados.

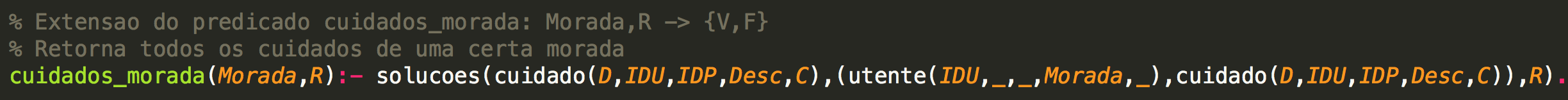
## Identificar cuidados de saúde prestados por instituição/cidade/data.

Neste ponto, o objetivo é identificar os cuidados de saúde por instituição, cidade e data.

Primeiro começamos por identificar todos os cuidados prestados por uma determinada instituição. Para isso, usou-se o ‘solucoes’ que procurou todos os identificadores dos prestadores da instituição dada como argumento e depois procurou todos os cuidados que tinham esses mesmos prestadores, retornando assim todos os cuidados em que os seus prestadores trabalham nessa instituição.

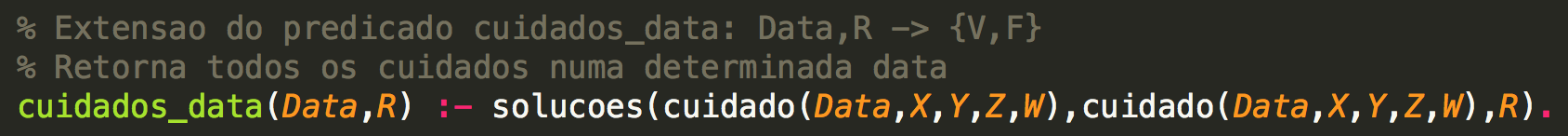
**Figura 25** - Resultado da ‘cuidados\_insti’ aplicada ao ‘Hospital de Santa Maria'.

**Figura 24** - Retorna os cuidados realizados por uma instituição.

Depois segue-se a identificação dos cuidados por cidade, no entanto, visto que nenhum possuía um atributo ‘Cidade’, utilizou-se a morada. A abordagem utilizada foi muito semelhante a anterior, pois procurou-se todos os utentes que possuíam aquela morada e retornou-se todos os cuidados que possuem esses utentes encontrados. Permitindo assim, que o resultado apresentado fosse todos os cuidados de uma morada especifica.

**Figura 27** - Resultado da ‘cuidados\_morada’ aplicada a 'Campos'.

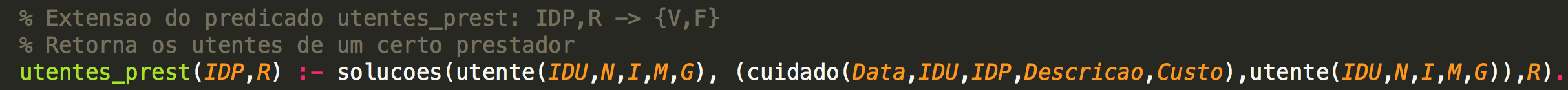
**Figura 26** - Cuidados obtidos de uma morada.

Por último, temos a função que identifica os cuidados prestados numa determianda data. Para isso, simplesmente usamos o ‘solucoes’ com a data pretendida e ele vai retornar a lista de todos os cuidados que foram realizados nessa determinada data, fornecida como argumento.

**Figura 29** - Resultado da ‘cuidados\_data’ aplicada ao dia 31-12-2018.

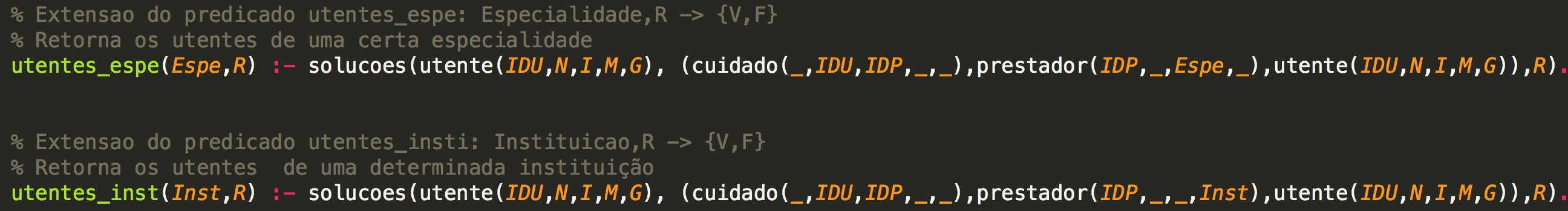
**Figura 28** - Cuidados realizados numa determinada data.

## Identificar os utentes de um prestador/especialidade/instituição

Para identificar todos os utentes de um certo prestador é necessário pesquisar nos cuidados e encontrar todos os identificadores de utentes que tinham sido tratados pelo prestador dado como argumento. Depois, apenas é necessário procurar os utentes que cumprem este requisito e retorná-los numa lista. Utilizando o ‘solucoes’, tal como apresentado em baixo, vai retornar todos os utentes de um prestador especifico.

**Figura 30** - Retorna os utentes de um prestador específico.

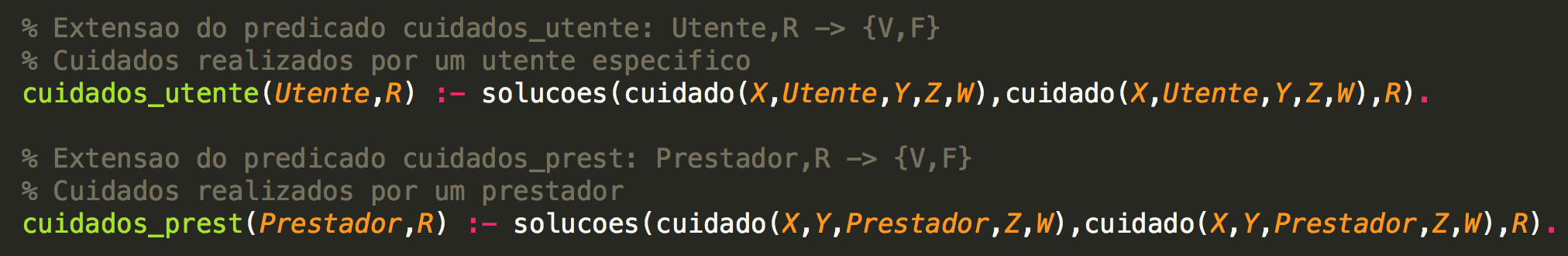
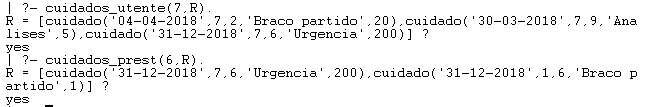
**Figura 31** - Resultado da ‘utentes\_prest’ aplicado ao prestador com ID=6.

Para os dois outros casos, a estrutura é a mesma, apenas alterando em que, em vez de se procurar nos cuidados e nos utentes, também tem de se procurar nos prestadores pela especialidade ou instituição pedida.

**Figura 32** – Predicados que retornam os utentes de uma certa especialidade e de uma certa instituição.

## Identificar cuidados de saúde realizados por utente/prestador

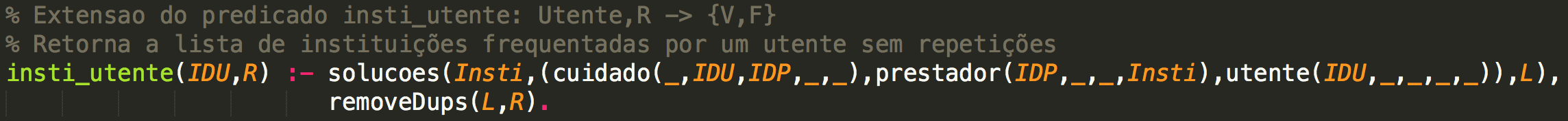
Para identificar os cuidados de saúde realizados por um utente basta procurar em todos os cuidados por esse utente e retornar apenas os cuidados que o incluem.

Para identificar os cuidados de um prestador a ideia é a mesma, excetuando que, em vez de procurar pelos utentes, procura-se antes pelo prestador, fornecido como argumento.

**Figura 34** - Resultado das funções

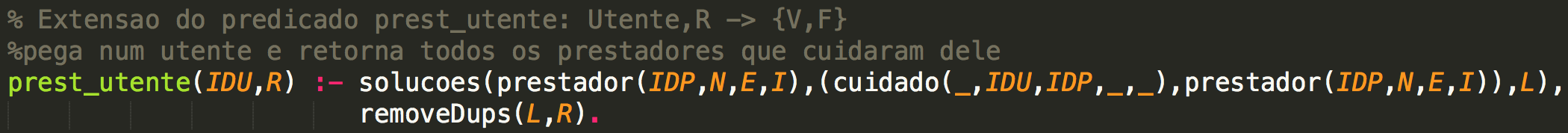
**Figura 33** – Funções que descobrem os cuidados realizados por um utente e por um prestador

## Determinar todas as instituições/prestadores a que um utente já recorreu

Para determinar todas as instituições que um utente já frequentou, é necessário ligar todos os conhecimentos, pois as instituições estão nos prestadores e para ligar prestadores a utentes é necessário usar os cuidados. Para isso, nos ‘solucoes’ forneceu-se os seguintes argumentos, nos utentes vamos buscar o utente dado com argumento, nos cuidados vamos buscar todos os cuidados que tenham o utente em causa e obtemos o identificador dos prestadores que cuidaram desse utente, e por fim, vamos aos prestadores obtidos e vamos retornar todas as instituições desses prestadores. Obtendo assim todas as instituições a que um utente recorreu, no entanto é necessário remover os repetidos, pois um utente pode ter realizado vários cuidados na mesma instituição que iram aparecer repetidamente.

**Figura 36** - Resultado de ‘insti\_utente(1, R)’.

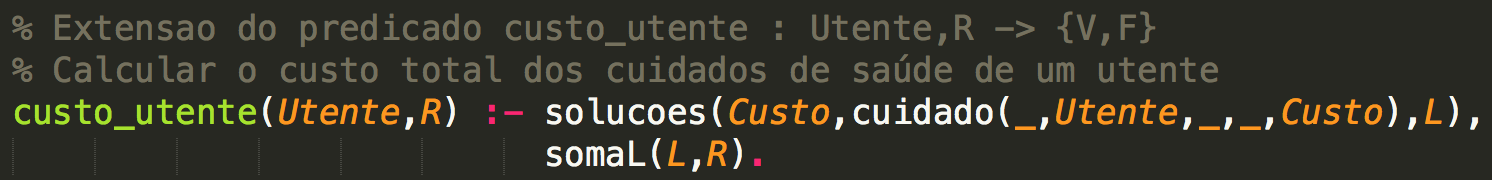
**Figura 35** - Retorna todas as instituições que um utente frequentou.

Para determinar todos os prestadores a que utente recorreu é necessário primeiro descobrir todos os identificadores de prestadores que lhe prestaram serviço, para isso, procura-se nos cuidados os que apresentam o identificador do utente dado e depois com o identificador do prestador obtido retorna-se as suas informações.

**Figura 37** - Resultado do prest\_utente(1, X)

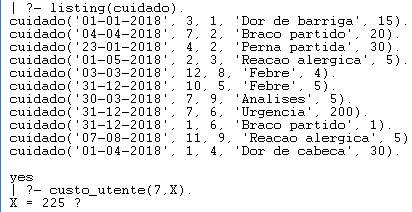
**Figura 38** - Retorna todos os prestadores de um utente

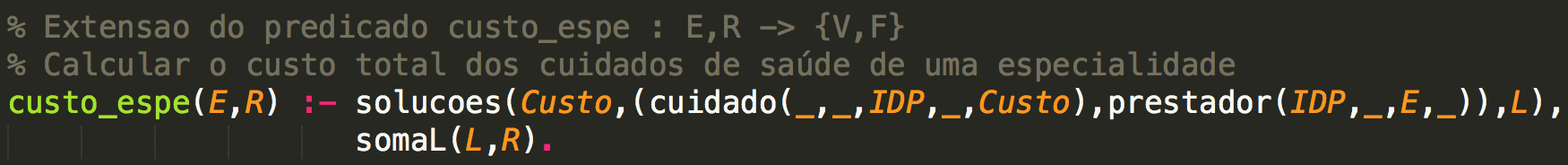
## Calcular o custo total dos cuidados de saúde por utente/especialidade/prestador/datas

Para calcular o custo total que um utente gastou é necessário procurar todos os cuidados em que esse utente aparece e guardar os seus respetivos custos numa tabela. Depois, com ajuda da função auxiliar ‘somaL’, que soma todos os elementos de uma lista, é só somar a lista de custos obtida e o resultado da função vai irá o resultado pretendido.

**Figura 40** - Resutado do custo\_utente(7, X)

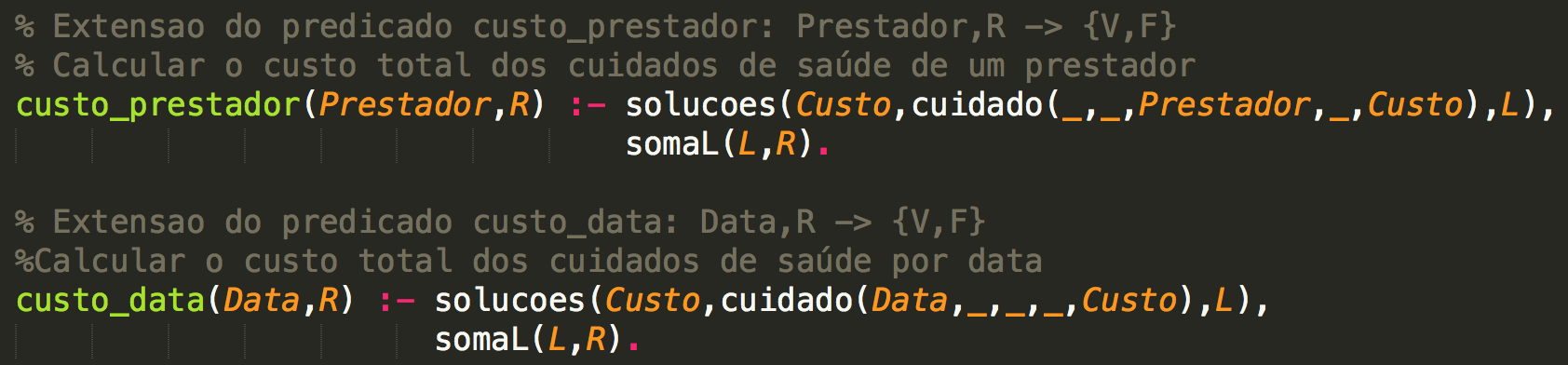
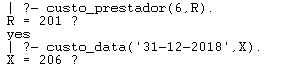
**Figura 39** - Determina o custo total de um utente



Para calcular o custo total de uma especialidade especifica é preciso ligar os prestadores e os cuidados, pois é neles que as informações estão contidas. Por isso, é necessário procurar pelos prestadores que apresentam a especialidade em causa e depois retornar os custos de todos os cuidados em que esses prestadores aparecem. Por fim, é só usar a função auxiliar ‘somaL’ para somar todos os valores da lista de custos e obtém-se o valor pretendido.

**Figura 42** - Resultado obtido

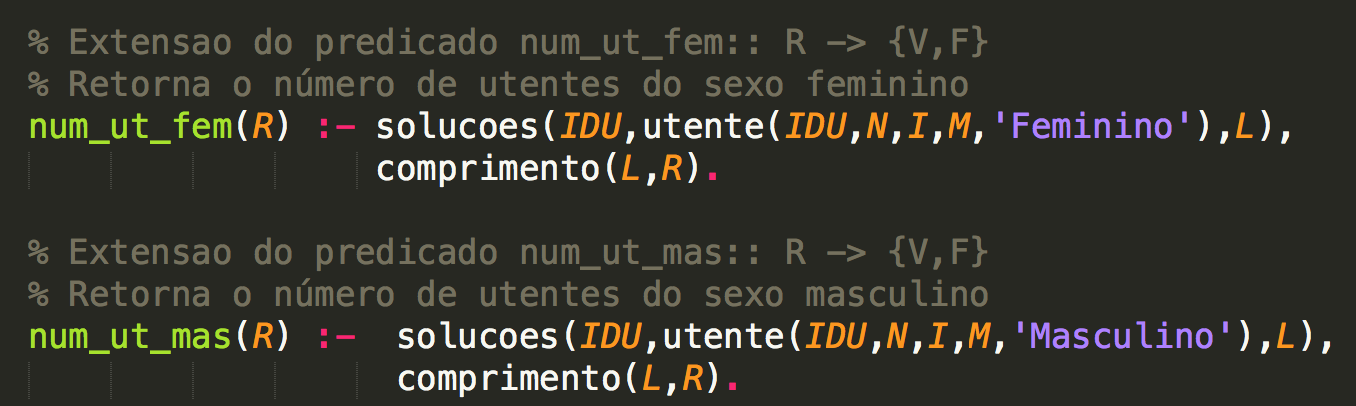
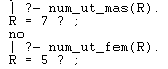
**Figura 41** - Obtém o custo total de uma certa especialidade.

Por último também se calculou os custos totais para um prestador e para uma determinada data.

**Figura 44** - Resultados obtidos.

**Figura 43** – Predicados que calculam o custo total para um prestador e para uma data.

# Extras

Para além dos exercícios propostos no enunciado ainda foram feitos alguns adicionalmente. Como por exemplo, o predicado capaz de calcular o número de utentes do sexo feminino e do sexo masculino. Para isso, procura todos os utentes masculinos ou femininos e depois é só retornar o tamanho dessa lista.

**Figura 46** - Resultado dos predicados.

**Figura 45** - Predicados que calculam o número de utentes do sexo feminino e masculino.

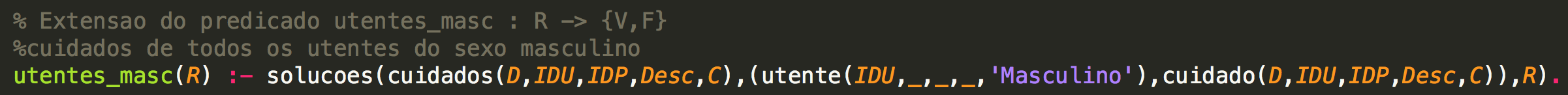
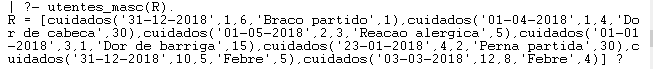
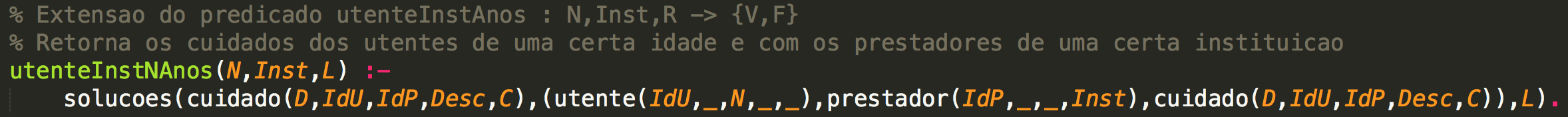
Também se criou um predicado que descobre todos os cuidados dos utentes do sexo masculino. Para isso, procura os identificadores dos utentes do sexo masculino e depois procura nos cuidados todos aqueles que tem este identificador retornando esses.

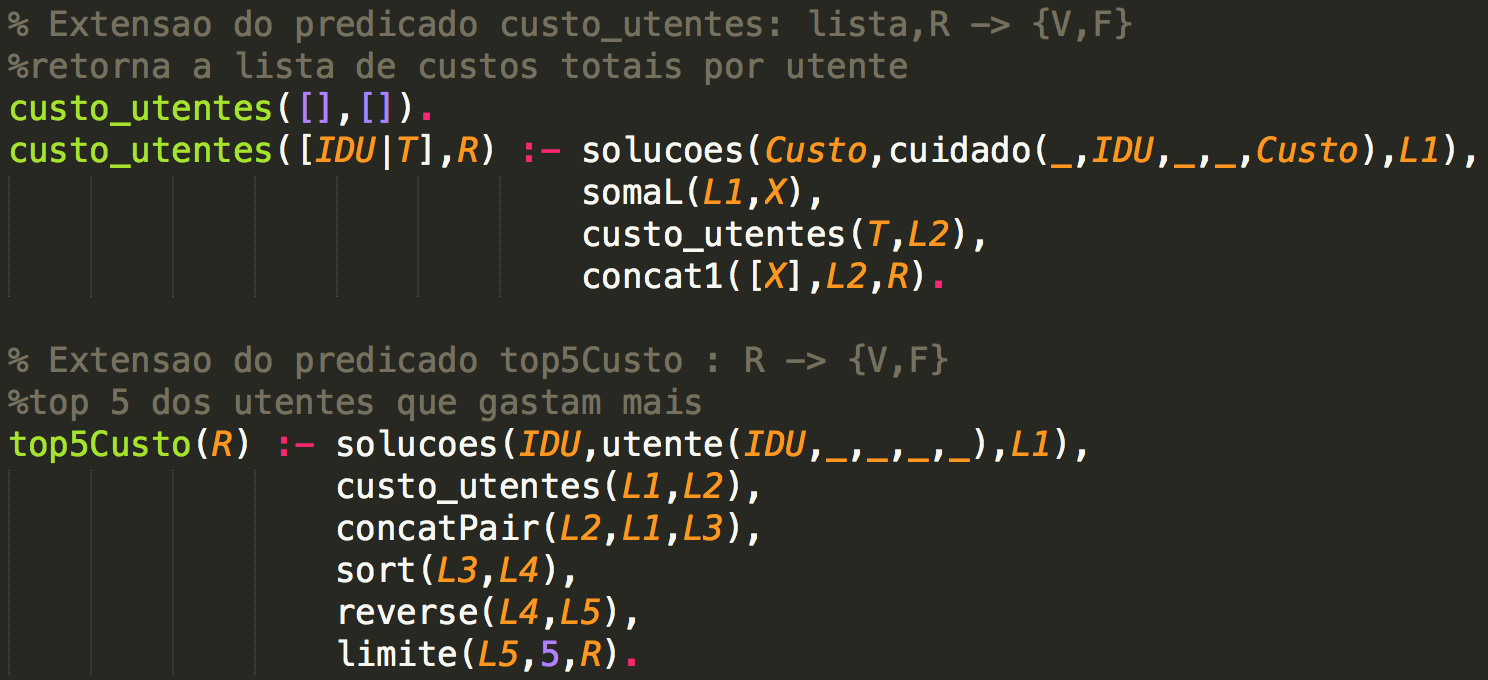
Figura 47 - Resultado da ‘utentes\_masc’.

Figura 48 - Cuidados de todos os utentes do sexo masculino.

Visto que, nenhum dos exercícios propostos misturava todos os conhecimentos, decidiu-se fazer uma função que retorna os cuidados de todos os utentes de uma certa idade e que foram tratados por prestadores de uma instituição especifica. Para isso, procura-se todos os utentes com a idade dada como argumento e procura-se também todos os prestadores da instituição, dada também como argumento da função, e depois é só preciso retornar todos os cuidados em que estes dois identificadores, tanto do utente como do prestador, estejam presentes.

**Figura 50** - Resultado da ‘utenteInstNanos(20, Hospital de Braga', R)’.

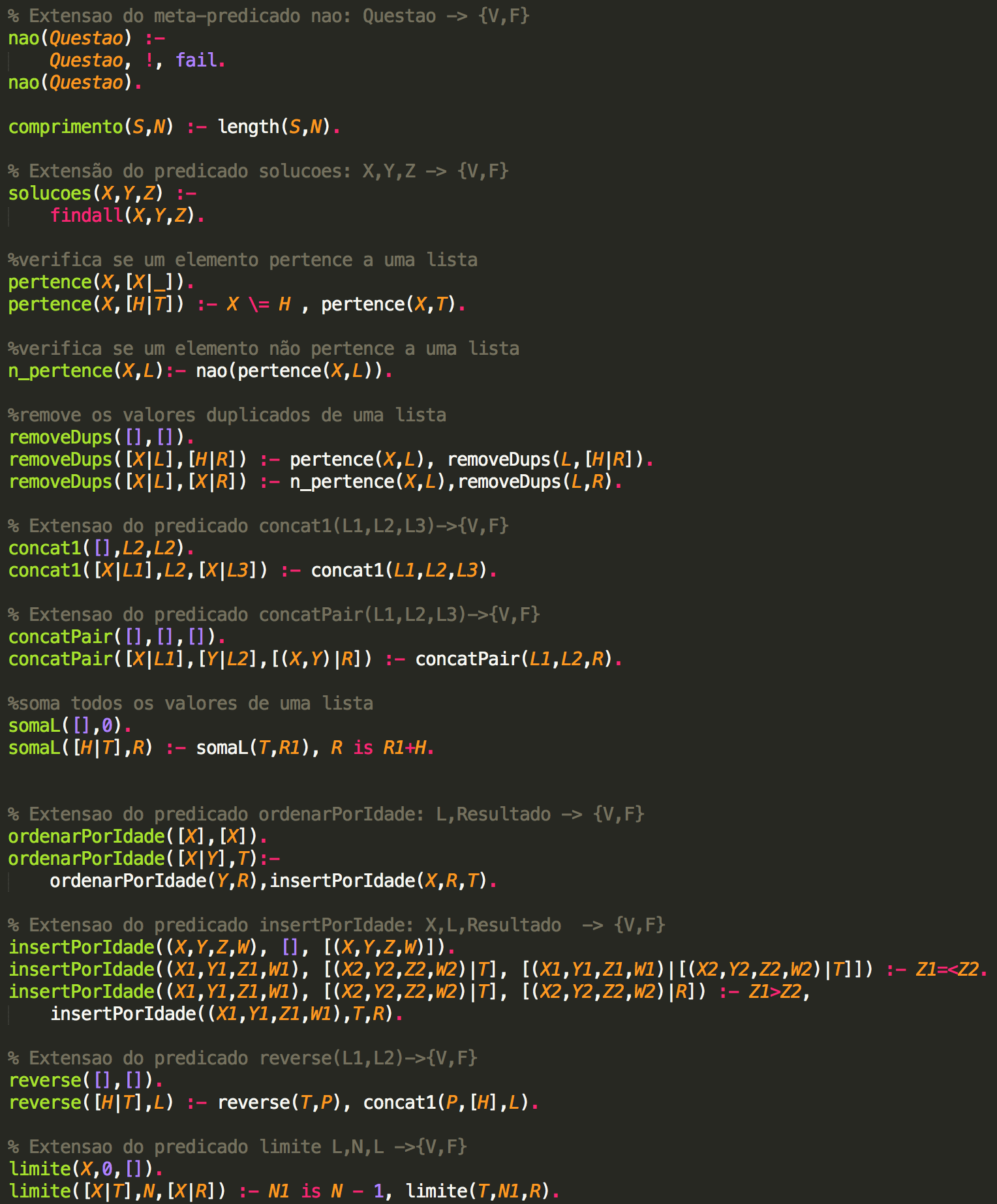
**Figura 49** - Descobre todos os cuidados dos utentes de uma certa idade e dos prestadores de uma certa instituição.

Por último, criou-se uma função capaz de descobrir o top 5 dos utentes que gastaram mais com cuidados de saúde. Para isso descobria-se a lista de custos totais ordenados por utentes, isto com ajuda da função ‘custo\_utentes’, depois ordenou-se crescentemente com a função ‘sort’, reverteu-se essa lista e por fim limitou-se a lista a 5 elementos. Retornado assim os cinco utentes que gastaram mais.

**Figura 52** - Resultado da top5Custo(R).

**Figura 51** - Função que retorna os 5 utentes que gastaram mais.

# Funções Auxiliares

Neste capítulo mostramos todas as funções auxiliares que foram usadas para a resolução de alguns algoritmos:

**Figura 53** - Funções auxiliares usadas.

# Conclusões e sugestões

Após concluir todos os exercícios propostos, podemos passar a uma análise final do trabalho realizado.

Podemos observar positivamente a realização deste trabalho, uma vez que concluímos todos os exercícios propostos corretamente, permitindo ainda solidificar conhecimentos aprendidos nas aulas e também adquirir novos conceitos acerca da programação lógica com o PROLOG.

Num projeto deste tipo, há quase sempre melhorias que podem ser implementadas, para permitir uma melhor manipulação do conhecimento guardado. Esse é sem dúvida um trabalho a desenvolver futuramente, que seria capaz de melhorar este sistema.