**Utente:**

Na nossa base de conhecimento o predicado utente representa um indivíduo ao qual é prestado um determinado cuidado, podemos então deduzir, que no mundo real o utente será um cidadão comum. Portanto tem de possuir um nome, mas como um nome não identifica inequivocamente uma pessoa numa atividade é necessário um identificador. Precisamos ainda de um campo com a idade dos cidadãos, DIZER PORQUE PRECISAMOS DA IDADE e também da sua morada para que possamos identificar os utentes de diferentes localidades. Para caraterizar a morada de um utente decidimos criar um predicado morada, sendo que esta contém um campo relativo à rua da morada do mesmo, outra relativo à localidade e outro relativo à cidade.

Pelas razões referidas anteriormente, decidimos criar o predicado:

FALTA POR O CONHECIMENTO QUE CADA UM MANIPULA

**utente( ID,Nome,Idade,morada( Rua,Localidade,Cidade ) )**

De seguida apresentamos alguns exemplos relativos a utentes presentes na base de conhecimento:

EXEMPLOS

**Invariantes relativos ao utente:**

Como referido anteriormente, o ID (campo) identifica inequivocamente um utente. Por isto não pode ser possível, aquando da inserção de novos utentes, inserir um utente com um identificador que já esteja presente na base de conhecimento. Decidimos criar um invariante para garantir esta questão. (Tanto é válido para o utente como para o prestador)

A manipulação do invariante foi feita da seguinte forma:

+utente( IdU,\_,\_,\_ ) :: (solucoes( IdU,utente( IdU,\_,\_,\_ ),L ),

comprimento( L,X ),

X =< 1).

Apenas temos de utilizar o predicado soluções (que nos dá todas as soluções para um predicado presentes na base de conhecimento), sendo que o predicado que procuramos são o utente com o identificador igual ao do utente adicionado. Depois apenas temos de garantir que a lista de soluções que obtivemos tem um comprimento inferior ou igual a 1 (poderia ser só igual).

Não é possível remover um utente da base de conhecimento para os quais ainda existem cuidados, visto que iria retirar consistência à mesma. Para garantir o referido, criamos um invariante. . (Tanto é válido para o utente como para o prestador)

A manipulação do invariante foi feita da seguinte forma:

-utente( IdU,\_,\_,\_ ) :: (solucoes( D,cuidado(D,IdU,\_,\_,\_,\_ ),L ),

comprimento( L,X ),

X == 0).

Utilizamos o predicado soluções para obter todas as datas dos cuidados para o utente com o identificador do utente que queremos retirar. Depois apenas podemos remover se o comprimento dessa lista for zero ou seja, caso não existam cuidados.

**Predicados de inserção:**

Para realizar os predicados de inserção, decidimos criar um predicado para cada um dos predicados referidos anteriormente.

Para isso necessitamos de criar um predicado geral que realiza uma evolução à nossa base de conhecimento, o predicado. Este predicado tem como função, testar a possibilidade de adicionar o facto referido atendendo aos invariantes criados e caso seja possível, então insere-o na base de conhecimento.

Decidimos criar o predicado **evolução( T )**, que manipula um facto como conhecimento. Este predicado guarda todos os invariantes relativos à inserção do facto que vai inserir numa lista, inserindo o facto e testando cada um dos invariantes após a inserção. Caso um dos invariantes falhe e não posso evoluir o conhecimento, então retira o facto inserido.

A inserção é realizada com um predicado auxiliar (**inserir**), que tem a responsabilidade de remover o facto previamente inserido caso os invariantes falhem.

O predicado de teste (**testar**) manipula como conhecimento um facto e uma lista de invariantes, sendo que terá de testar cada um destes invariantes para o facto referido. Apenas será verdadeiro caso todos os invariantes sejam positivamente testados.

De seguida apresentamos os predicados referidos:

FALTA POR O CONHECIMENTO QUE CADA UM MANIPULA

evolucao( Termo ) :- solucoes( Inv,+Termo::Inv,S ),

inserir( Termo ),

testar( S ).

inserir( P ) :- assert( P ).

inserir( P ) :- retract( P ), !, fail.

testar( [] ).

testar( [X|R] ) :- X,

testar( R ).

Para realizar a inserção de um utente criamos o predicado adicionarUtente que manipula exatamente o mesmo conhecimente que o predicado utente. Este predicado auxilia-se no predicado evolução, passando como termo para evoluir o conhecimento o utente com o conhecimento recebido.

De seguida apresentamos o predicado referido:

adicionarUtente: Id,Nome,Idade,Morada -> {V,F}

adicionarUtente( IdUtente,Nome,Idade,Morada ) :- evolucao( utente( IdUtente,Nome,Idade,Morada ) ).

Os restantes predicados de inserção, para prestadores, cuidados e instituições seguem o mesmo raciocínio que este pelo que não aprofundamos a sua explicação, referindo-os em anexo.

**Predicados de remoção:**

Para realizar os predicados de remoção utilizamos o mesmo raciocínio dos predicados de inserção. A única diferença é que neste caso queremos fazer uma involução à base de conhecimento, pelo que criamos o predicado involução. Para além disto, este terá de guardar uma lista com os invariantes relativos à remoção do facto que será removido e removerá este facto caso passe nos testes realizados a todos os invariantes.

Para isso utilizamos os seguintes predicados:

FALTA POR O CONHECIMENTO QUE CADA UM MANIPULA

involucao( Termo ) :- solucoes(Inv,-Termo::Inv,S),

remover(Termo),

testar(S).

remover( P ) :- retract( P ).

remover( P ) :- assert( P ), !, fail.

Para realizar a inserção de um utente criamos o predicado retirararUtente que manipula como conhecimento o identificador de um utente, pois este será suficiente para o identificar e posteriormente remover. Este predicado auxilia-se no predicado involução, passando como termo para involuir o conhecimento o utente com o conhecimento recebido, e com os restantes campos pertencentes ao predicado utente como quaisquer visto que o mecanismo de teste do *Prolog* irá encontrar o único utente com aquele id, caso exista.

Os restantes predicados de remoção, para prestadores e instituições seguem o mesmo raciocínio que este pelo que não aprofundamos a sua explicação, referindo-os em anexo.

Para remover um cuidado, e como consideramos que este apenas não poderia ser igual a outro existente na base de conhecimento, é necessário que o predicado de remoção do mesmo manipule toda a informação que o predicado cuidado manipula. Por isto, criamos o seguinte predicado:

retirarCuidado: Data,IdUtente,IdPrestador,Descricao,Custo,Instituiao -> {V,F}

retirarCuidado( Data,IdUtente,IdPrestador,Descricao,Custo,Instituiao ) :- inevolucao( cuidado( Data,IdUtente,IdPrestador,Descricao,Custo,Instituiao ) ).