Universidade do Minho

Licenciatura em Engenharia Biomédica

Programação em lógica estendida e Conhecimento imperfeito

Inteligência Artificial em Engenharia Biomédica

Ano Letivo 2025/2026

Ana Filipa Figueiredo - a107239

Dinis Rosa - a107159

Duarte Franco – a107233

Resumo

A elaboração deste trabalho prático teve como principal objetivo o desenvolvimento de um Sistema de Representação de Conhecimento e Raciocínio, utilizando a linguagem *Prolog* e a Extensão à Programação em Lógica, no âmbito da representação de Conhecimento Imperfeito. O universo de discurso escolhido incidiu sobre a prestação de cuidados de saúde, com foco no acompanhamento e registo de pacientes e suas consultas de tensão arterial. Para tal, o sistema foi construído de forma a demonstrar a representação de Conhecimento Perfeito (positivo e negativo) e Conhecimento Imperfeito (incerto, impreciso e interdito) recorrendo à utilização de valores nulos.

O sistema incluiu a criação de Invariantes para estabelecer restrições à inserção e remoção de conhecimento, garantindo a integridade da base de dados. Foram também implementados mecanismos de Evolução e Involução do Conhecimento para gerir a atualização da base de dados. Um Sistema de Inferência (SI) foi desenvolvido, formalizado através do meta-predicado *si/2*, capaz de responder com os valores Verdadeiro ($\mathbb{V}$) Falso ($\mathbb{F}$) ou Desconhecido ($\mathbb{D}$). Adicionalmente, o trabalho integrou funcionalidades para classificar a tensão arterial dos pacientes e atualizar o seu diagnóstico.

Em suma, este exercício permitiu a consolidação de conhecimentos teóricos e práticos na área da Inteligência Artificial Simbólica e programação em lógica, resultando num sistema coerente e lógico no contexto clínico.

1. Introdução

No âmbito da unidade curricular Inteligência Artificial em Engenharia Biomédica, integrada no 3.º ano, 1.º semestre da Licenciatura em Engenharia Biomédica da Universidade do Minho, foi-nos proposto o desenvolvimento da primeira parte de um trabalho de grupo dedicado à aplicação prática de técnicas de Inteligência Artificial Simbólica. Esta componente prática pretende consolidar os conhecimentos adquiridos nas aulas, através da construção de um sistema de representação de conhecimento e raciocínio, suportado pela extensão à programação em lógica e pela utilização da linguagem PROLOG.

O exercício proposto consiste na criação de um sistema capaz de caracterizar um universo de discurso na área da saúde, mais concretamente no acompanhamento e registo da tensão arterial. Este domínio, pela sua relevância clínica e pela natureza frequentemente incompleta ou incerta dos dados fisiológicos, constitui um contexto adequado para explorar os desafios associados à representação de conhecimento imperfeito. Assim, recorre-se à utilização de valores nulos de diferentes tipos e à construção de mecanismos de raciocínio capazes de distinguir entre conhecimento Verdadeiro ($\mathbb{V}$), Falso ($\mathbb{F}$) e Desconhecido ($\mathbb{D}$).

O sistema a desenvolver deverá ser capaz de representar entidades fundamentais do cenário, nomeadamente paciente, consulta e tensão arterial, assegurando não só a representação de conhecimento positivo e negativo, mas também a modelação rigorosa de situações com incerteza ou informação incompleta. Para garantir a consistência interna da base de conhecimento, torna-se igualmente necessária a definição de invariantes que restrinjam a inserção e remoção de conhecimento, bem como a implementação de procedimentos que permitam gerir corretamente a evolução e involução do conhecimento.

Além disso, pretende-se demonstrar a capacidade do sistema de implementar mecanismos de inferência lógica, permitindo extrair conclusões sobre a condição médica dos pacientes com base nos registos e classificações de tensão arterial. Com este propósito, o caso prático desenvolvido deverá ilustrar, de forma clara, a utilidade da programação em lógica estendida no tratamento de conhecimento imperfeito e na modelação de sistemas biomédicos.

A realização deste trabalho não se limita à aplicação mecânica de técnicas teóricas; representa também um passo importante para o desenvolvimento de competências práticas essenciais à construção de sistemas inteligentes na área da saúde. Assim, esta primeira parte do relatório apresenta o problema proposto, os objetivos estabelecidos, os elementos de conhecimento modelados, bem como a estrutura conceptual e técnica do sistema de representação de conhecimento desenvolvido.

2 *Preliminares*

A Programação em Lógica é baseada na lógica formal, onde problemas são descritos por Cláusulas que podem ser de 3 tipos, factos, regras e consultas. Ao invés de instruir passo a passo, o programador define o que é verdadeiro, e o sistema deduz soluções automaticamente.

O PROLOG é a linguagem mais conhecida desse paradigma, especializada em inferência lógica e manipulação de relações. Ele permite representar conhecimento e realizar deduções complexas de forma declarativa, sendo muito utilizado na área da Inteligência Artificial.

2.1 Cláusulas

2.1.1 Factos

Os factos representam verdades absolutas sobre o domínio do problema e são expressos de forma simples, como por exemplo:

(“O paciente com ID 1 teve uma consulta no dia 12 de março de 2024, com o ID de consulta de 1001”)

```
consulta(1001, 1, date(12, 3, 2024), nulo_val, 118, 76, 72, realizada).
```

Figura 1: Exemplo de um facto em prolog

2.1.2 Regras

As regras definem relações condicionais, sendo verdadeiras apenas se determinadas condições forem cumpridas como por exemplo:

(“O paciente com ID 1 teve uma consulta no dia 12 de março de 2024, se o ID de consulta for 1001”)

```
consulta(1001, (12,3,2022),realizada) :- ID_consulta(1001)
```

Figura 2: Exemplo de uma regra condicional

2.1.3 Questões

No Prolog, as questões são expressões utilizadas para verificar se determinadas afirmações são verdadeiras ou falsas, com base nas condições e no conhecimento previamente definido na base de dados.

(“A consulta de ID 1001 do paciente de ID 1 para o dia 12 de março de 2024 foi realizada?”)

```
?- consulta(1001, 1, date(12,3,2024), nulo_val, 118, 76, 72, realizada).
true.
```

Figura 3: Exemplo de uma questão simples

O prolog responde a esta pergunta com true ou false, dependendo assim se a informação dada é verdadeira ou não.

2.2 Pressuposto do Mundo Fechado (PMF)

O Pressuposto do Mundo Fechado assume que a base de conhecimento contém toda a informação relevante sobre o domínio. Assim, qualquer facto que não esteja registado é automaticamente tratado como falso. Por exemplo, Filipa (ID 13) não é paciente da clínica. E assumimos isso como verdade, porque não existe qualquer registo dela nem nenhuma exceção que indique que o caso é especial

```
-paciente(13, filipa, date(30, 4, 2005), f, braga, 173, 71, nulo_val) :-
    nao(paciente(13, filipa, date(30, 4, 2005), f, braga, 173, 71, nulo_val)),
    nao(excecao(paciente(13, filipa, date(30, 4, 2005), f, braga, 173, 71, nulo_val))).
```

Figura 4: Formalização do PMF em Prolog

Considera-se falso que exista este paciente, *desde que* não haja prova do contrário nem exceção que permita considerá-lo possível.

2.3 Programação em Lógica (PL)

A Programação em Lógica (PL) é um paradigma de programação declarativa que utiliza a lógica de primeira ordem (especificamente, Cláusulas de *Horn*) para representar o conhecimento sobre um problema e para realizar o raciocínio.

Em vez de se focar em como resolver um problema (como na programação imperativa), a Programação em Lógica foca-se no que é o problema, ou seja, na sua descrição lógica.

2.3.1 Pressupostos da PL

Todas as bases de dados computacionais necessitam de ter a capacidade de armazenar e manipular informação. É neste espeto que as linguagens de manipulação de informação, tal como o *Prolog*, ou outras linguagens de programação em lógica são importantes.

Estas, devem ter alicerces em diferentes pressupostos, tais como o Pressuposto do Mundo Fechado (PMF), onde toda a informação que não está mencionada na base de dados é considerada falsa.

No entanto, no âmbito deste trabalho que visa a representação de conhecimento imperfeito (englobando conhecimento incerto, impreciso e interdito), foi adotado o Pressuposto do Mundo Aberto (PMA). O PMA implica que as questões feitas à base de conhecimento devem ter um contradomínio de três respostas possíveis: verdadeiro, falso ou desconhecido.

2.3.2 Extensão à PL

A Extensão à Programação em Lógica (EPL) tem como objetivo primordial permitir a representação explícita do conhecimento negativo. Este conhecimento negativo pode ser abordado através de dois tipos de negação. A primeira é a negação por falha na prova (representada pelo predicado *nao*), na qual o *Prolog* infere que uma proposição é falsa se falhar em provar que é verdadeira (baseado no Pressuposto do Mundo Fechado). Contudo, há situações em que esta dependência se torna inaceitável, bastando que a informação não esteja contemplada para que o sistema assuma a sua falsidade, mesmo que a verdade seja desconhecida ou que se saiba explicitamente que é falsa.

Para superar esta limitação e possibilitar a representação do conhecimento negativo assertivo (ou conhecimento perfeito negativo), a EPL introduz a negação forte (ou clássica), que utiliza o prefixo "-". Esta negação permite afirmar inequivocamente que uma dada informação é falsa, o que é crucial para ter a possibilidade de representar corretamente o conhecimento negativo e tomar decisões mais fiáveis sobre a existência do mesmo na base de conhecimento.

1. Verdadeira (V) — há prova do facto.

2. Falsa (F) — há prova de que o facto não é verdadeiro.

3. Desconhecida (D) — não existe conhecimento suficiente.

Além disso, são utilizados dois tipos de negação:

- Negação por falha — usada para representar a ausência de prova.
- Negação forte — usada para afirmar explicitamente que algo é falso.

```
nao(Q) :- Q, !, fail.
nao(_).

-paciente(Id, Nome, Data, Sexo, Morada, Altura, Peso, Hist) :-
    nao(paciente(Id, Nome, Data, Sexo, Morada, Altura, Peso, Hist)),
    nao(excecao(paciente(Id, Nome, Data, Sexo, Morada, Altura, Peso, Hist))).
```

Figura 5: Negação por falha e negação forte

2.4 Invariantes da Inserção

Os invariantes são restrições que asseguram a consistência da base de conhecimento. Podem ser classificados em:

- Estruturais — impedem a duplicação de dados;
- Referenciais — garantem a coerência entre os valores registados.

2.5 Valores Nulos e Conhecimento Imperfeito

Para este trabalho, foi fundamental representar o conhecimento imperfeito. Além dos valores binários verdadeiro e falso, tornou-se necessária a introdução de um terceiro valor: desconhecido.

Essa representação é utilizada em situações onde a informação está incompleta. Para lidar com essas lacunas, recorremos a valores nulos, que permitem ao sistema diferenciar quando uma resposta é conhecida (verdadeiro ou falso) ou desconhecida.

Os valores nulos (“nulo_val”) podem ser classificados em três tipos:

- **Incerto:** O valor possui infinitas possibilidades, podendo ser definido posteriormente;
- **Impreciso:** O valor é desconhecido, mas pertence a um conjunto finito de hipóteses;
- **Interdito:** O valor não pode ser definido nem conhecido, apresentando infinitas possibilidades.

3 Descrição do trabalho e Análise de Resultados

3.1 Flags

No início do programa, são definidas as chamadas flags, cujo propósito principal é prevenir a emissão de avisos pelo interpretador Prolog. Estas configurações podem ser ajustadas conforme as necessidades do utilizador, permitindo um maior controlo sobre o comportamento do sistema.

```
:- set_prolog_flag(discontiguous_warnings, off).
:- set_prolog_flag(single_var_warnings, off).
```

Figura 6: Declarações iniciais.

A instrução set_prolog_flag(discontinuous_warnings, off) tem como função inibir alertas relacionados com predicados descontínuos, ou seja, aqueles cuja definição é interrompida por outras declarações de predicados no mesmo programa. De forma semelhante, a diretiva set_prolog_flag(single_var_warning, off) serve para suprimir avisos referentes a variáveis únicas, o que é útil quando a utilização de tais variáveis é intencional, evitando notificações desnecessárias durante a execução do código.

A adoção destas configurações contribui para um ambiente de execução mais limpo e controlado, permitindo que o utilizador se concentre na lógica e funcionalidade do programa sem ser distraído por alertas que não comprometem o correto funcionamento do sistema.

3.2 Dynamic

Todos os predicados devem ser declarados como dynamic, permitindo que factos que seriam, por defeito, estáticos possam ser alterados ao longo da execução do programa. Esta abordagem torna a base de conhecimento dinâmica, possibilitando a adição ou remoção de informação associada aos predicados definidos, conferindo maior flexibilidade, capacidade de adaptação ao sistema e suporte à evolução do conhecimento ao longo do tempo. Esta característica é particularmente relevante em contextos onde os dados estão sujeitos a alterações frequentes ou provêm de múltiplas fontes.

```
:- dynamic('-'/1).
:- dynamic paciente/8.
:- dynamic consulta/8.
:- dynamic tensao_arterial_negativa/6.
:- dynamic excecao/1.
:- dynamic interdito/1.
:- dynamic si/2.
```

Figura 7: Definições iniciais.

3.3 Operadores

Foram introduzidos operadores específicos, como o '::', que possibilitam a definição de invariantes nos processos de inserção e remoção de conhecimento. Estes invariantes asseguram que a integridade da base de dados seja preservada durante as operações de atualização, prevenindo inconsistências, duplicações ou a introdução de informações inválidas. Desta forma, o sistema consegue manter a coerência lógica da base de conhecimento, mesmo quando sujeitos a alterações dinâmicas, garantindo fiabilidade e robustez nas operações de consulta e dedução.

3.4 Base de Conhecimento

A Base de Conhecimento é constituída por diferentes entidades, sendo elas o paciente, a consulta e a tensão arterial.

3.4.1 Paciente

Na Base de Conhecimento o paciente é caracterizado pelos seguintes parâmetros:

- ID do Paciente: número inteiro único;
- Nome do paciente;
- Data de Nascimento: do tipo date(dd, mm, aaaa);
- Sexo: Masculino (m) ou Feminino (f);
- Distrito (ex: Coimbra);
- Altura, em centímetros (ex: 165);
- Peso, em quilogramas (ex: 65.5);
- Histórico de Diagnósticos: lista de pares [dataConsulta, Diagnostico] ou valor nulo (*nulo_val*).

```
% Pacientes Femininos (f)
paciente(1, maria, date(20, 5, 1985), f, braga, 165, 62.5, nulo_val).
paciente(2, ana, date(15, 8, 2001), f, porto, 158, 55.2, nulo_val).
paciente(3, joana, date(1, 4, 2010), f, lisboa, 140, 40.0, nulo_val).
paciente(4, luisa, date(19, 1, 1953), f, coimbra, 160, 70.3, nulo_val).
paciente(5, nadia, date(5, 12, 2005), f, faro, 150, 48.6, nulo_val).

% Pacientes Masculinos (m)
paciente(6, joao, date(3, 11, 1972), m, aveiro, 178, 85.0, nulo_val).
paciente(7, jose, date(28, 2, 1960), m, setubal, 172, 92.1, nulo_val).
paciente(8, antonio, date(10, 9, 1995), m, viseu, 185, 75.8, nulo_val).
paciente(9, manuel, date(25, 7, 1988), m, leiria, 175, 79.9, nulo_val).
paciente(10, carlos, date(17, 6, 1977), m, santarem, 180, 88.7, nulo_val).
```

Figura 8: Base de Conhecimento dos Pacientes

3.4.2 Consulta

As consultas são definidas por:

- ID da Consulta: número inteiro único;
- ID do Paciente: número inteiro único;
- Data da Consulta: do tipo date(dd, mm, aaaa);
- Idade do Paciente, número inteiro ou valor nulo (*nulo_val*);
- Medição do valor da Sistólica, em mmHg (ex: 98);
- Medição do valor da Diastólica, em mmHg (ex: 70);
- Frequência Cardíaca, em bpm (ex: 100);
- Estado: *agendada*, *realizada* ou *cancelada*.

```
% Consultas Realizadas
consulta(1001, 1, date(12, 3, 2024), nulo_val, 118, 76, 72, realizada).
consulta(1002, 2, date(5, 4, 2024), nulo_val, 110, 70, 68, realizada).
consulta(1003, 3, date(22, 5, 2024), nulo_val, 102, 66, 85, realizada).
consulta(1004, 4, date(14, 6, 2024), nulo_val, 130, 84, 78, realizada).
consulta(1005, 5, date(2, 7, 2024), nulo_val, 115, 72, 74, realizada).

% Consultas Agendadas (sem medições)
consulta(1006, 6, date(19, 8, 2025), nulo_val, nulo_val, nulo_val, nulo_val, agendada).
consulta(1007, 7, date(9, 9, 2025), nulo_val, nulo_val, nulo_val, nulo_val, agendada).
consulta(1008, 8, date(1, 10, 2025), nulo_val, nulo_val, nulo_val, nulo_val, agendada).

% Consultas Canceladas (sem medições)
consulta(1009, 9, date(20, 11, 2024), nulo_val, nulo_val, nulo_val, nulo_val, cancelada).
consulta(1010, 10, date(12, 12, 2024), nulo_val, nulo_val, nulo_val, nulo_val, cancelada).
```

Figura 9: Base de Conhecimento das consultas

3.4.3 Tensão Arterial

Cada tensão arterial é imutável e está associada a um intervalo de valores de Sistólica e Diastólica e a uma classificação. Esta serve apenas de *lookup table*, para determinação do diagnostico do paciente. É definida por:

- ID da Tensão: número inteiro único;
- Classificação: grau da tensão arterial;
- Valor de Sistólica inferior, em mmHg (ex: 0);
- Valor de Sistólica superior, em mmHg (ex: 119) ou infinito (*sem_limite*);
- Valor de Diastólica inferior, em mmHg (ex: 0)
- Valor de Diastólica superior, em mmHg (ex: 79) ou infinito (*sem_limite*);

```
% Tensão Arterial - Classificações
tensao_arterial(2001, otima,                        0, 119,   0,  79).
tensao_arterial(2002, normal,                     120, 129,  80,  84).
tensao_arterial(2003, normal_alta,                130, 139,  85,  89).
tensao_arterial(2004, hipertensao_grau1,          140, 159,  90,  99).
tensao_arterial(2005, hipertensao_grau2,          160, 179, 100, 109).
tensao_arterial(2006, hipertensao_grau3,          180, sem_limite, 110, sem_limite).
tensao_arterial(2007, hipertensao_sistolica_isolada, 140, sem_limite, 0, 89).
```

Figura 10: Base de Conhecimento da tensão arterial

3.5 Programação em Lógica Estendida

No contexto da lógica estendida implementada, todos estes predicados podem assumir valores do conjunto {V, F, D}, permitindo ao sistema diferenciar entre conhecimento verdadeiro, falso ou desconhecido. Esta capacidade é fundamental para lidar com situações de conhecimento incompleto ou imperfeito, refletindo de forma mais precisa a realidade, onde nem toda a informação é disponível ou confiável.

Além disso, a utilização de valores nulos e a distinção entre conhecimento positivo e negativo permitem realizar inferências mais robustas e desenvolver mecanismos de controlo de integridade que consideram a incerteza. Esta abordagem facilita também a implementação de sistemas de apoio à decisão, permitindo analisar padrões, detectar inconsistências e fornecer alertas quando determinados valores ou combinações de dados não se enquadram nos parâmetros esperados.

3.6 Sistema de Inferência

A gestão de informação incompleta ou incerta constitui um dos maiores desafios em sistemas de inteligência artificial. Para lidar com estas situações, é necessário que os sistemas de inferência considerem não apenas os valores verdadeiro e falso, mas também o valor desconhecido, permitindo representar de forma precisa casos em que a informação disponível é insuficiente para determinar a veracidade de um facto.

Com base nesta premissa, foi inicialmente desenvolvido um sistema de inferência básico (SI), concebido para avaliar questões simples. Este sistema é capaz de determinar a validade de um facto individual, retornando como resultado verdadeiro, falso ou desconhecido, de acordo com o conhecimento presente na base de dados. O SI constitui, portanto, a infraestrutura inicial para a realização de inferências sobre o conhecimento armazenado.

```
si( Questao, verdadeiro):- Questao.
si( Questao, falso):- -Questao.
si( Questao, desconhecido) :- nao(Questao), nao(-Questao).
```

Figura 11: Sistema de inferencia

No entanto, em contextos mais complexos, onde se pretende analisar relações entre múltiplos factos, torna-se necessário recorrer a sistemas capazes de processar operações lógicas compostas. Para satisfazer esta necessidade, foram desenvolvidos dois sistemas adicionais de inferência:

- **SIC (Sistema de Inferência por Conjunção):** permite avaliar conjunções de factos, determinando se todas as condições de um conjunto de questões são satisfeitas simultaneamente. Este mecanismo é fundamental quando a validade de um resultado depende da verificação simultânea de múltiplas afirmações.
- **SID (Sistema de Inferência por Disjunção):** permite avaliar disjunções de factos, verificando se pelo menos uma das condições consideradas é verdadeira. Este sistema é útil em situações em que basta que uma das afirmações seja satisfeita para que o resultado global seja considerado verdadeiro.

```
siC( Q1, Q2, verdadeiro) :-
    si(Q1, verdadeiro),
    si(Q2, verdadeiro).
siC( Q1, Q2, falso) :-
    si(Q1, verdadeiro),
    si(Q2, falso).
siC( Q1, Q2, desconhecido) :-
    si( Q1, verdadeiro),
    si( Q2, desconhecido).
siC( Q1, Q2, falso) :-
    si( Q1, falso),
    si( Q2, verdadeiro).
siC( Q1, Q2, falso) :-
    si( Q1, falso),
    si( Q2, falso).
siC( Q1, Q2, falso) :-
    si( Q1, falso),
    si( Q2, desconhecido).
siC( Q1, Q2, desconhecido) :-
    si( Q1, desconhecido),
    si( Q2, verdadeiro).
siC( Q1, Q2, falso) :-
    si( Q1, desconhecido),
    si( Q2, falso).
siC( Q1, Q2, desconhecido) :-
    si( Q1, desconhecido),
    si( Q2, desconhecido).
```

Figura 12: Sistema de inferência de conjunção

```
siD( Q1, Q2, verdadeiro) :-
    si( Q1, verdadeiro),
    si( Q2, verdadeiro).
siD( Q1, Q2, verdadeiro) :-
    si( Q1, verdadeiro),
    si( Q2, falso).
siD( Q1, Q2, verdadeiro) :-
    si( Q1, verdadeiro),
    si( Q2, desconhecido).
siD( Q1, Q2, verdadeiro) :-
    si( Q1, falso),
    si( Q2, verdadeiro).
siD( Q1, Q2, falso) :-
    si( Q1, falso),
    si( Q2, falso).
siD( Q1, Q2, desconhecido) :-
    si( Q1, falso),
    si( Q2, desconhecido).
siD( Q1, Q2, verdadeiro) :-
    si( Q1, desconhecido),
    si( Q2, verdadeiro).
siD( Q1, Q2, desconhecido) :-
    si( Q1, desconhecido),
    si( Q2, falso).
siD( Q1, Q2, desconhecido) :-
    si( Q1, desconhecido),
    si( Q2, desconhecido).
```

Figura 13: Sistema de inferência de disjunção

A implementação destes sistemas avançados confere ao modelo de inferência uma capacidade de raciocínio mais sofisticada, permitindo que o sistema lide com diferentes combinações de factos e condições, mesmo na presença de conhecimento incompleto.

A inclusão do valor desconhecido garante a robustez das inferências, evitando conclusões precipitadas ou incorretas e assegurando uma interpretação rigorosa da informação disponível.

Deste modo, os sistemas SI, SIC e SID constituem uma estrutura integrada de inferência lógica, capaz de suportar aplicações de Inteligência Artificial em domínios que exigem análise complexa de dados, avaliação de múltiplas condições e apoio à decisão, como é o caso de sistemas clínicos, de planeamento ou de diagnóstico automatizado.

3.7 Predicados auxiliares

Nesta secção procede-se à apresentação dos predicados que se revelaram essenciais para a implementação das diversas funcionalidades do sistema.

3.7.1 Predicado comprimento

O predicado comprimento calcula o número de elementos de uma lista. O caso base diz que a lista vazia tem comprimento 0, e o caso recursivo conta 1 para o primeiro elemento e chama-se sobre o resto da lista, somando 1 a cada passo até chegar ao final.

```
comprimento([],0).
comprimento([_|L],N):-
   comprimento(L, N1),
   N is N1 + 1.
```

Figura 14: Predicado comprimento

3.7.2 Meta-predicado não

O predicado nao/1 implementa a negação em Prolog. Ele tenta provar Q; se conseguir (Q é verdadeiro), usa ! para cortar outras alternativas e falha com fail, indicando que nao(Q) é falso. Se Q não puder ser provado, então nao(_) é verdadeiro, ou seja, nao(Q) é considerado verdadeiro quando Q não pode ser provado.

```
nao(Q) :- Q, !, fail.
nao(_).
```

Figura 15: Meta-predicado não

3.7.3 Predicado pertence

O predicado pertence verifica se um elemento X está presente numa lista. Na primeira regra, X é o mesmo que o primeiro elemento da lista, então pertence(X, [X|_]) é verdadeiro. Na segunda regra, se X não for o primeiro elemento, a verificação continua recursivamente no resto da lista T com pertence(X, T). Assim, o predicado percorre a lista até encontrar X ou chegar ao fim.

```
pertence(X, [X|_]).
pertence(X, [_|T]) :-
    pertence(X, T).
```

Figura 16: Predicado pertence

3.7.4 Predicado teste

O predicado teste/1 serve para verificar se todos os elementos de uma lista satisfazem uma determinada condição. Ele considera que uma lista vazia é sempre verdadeira. Para listas não vazias, verifica se o primeiro elemento é verdadeiro e, em seguida, aplica-se recursivamente ao resto da lista, garantindo que todos os elementos sejam validados.

```
teste([]).
teste([R|LR]) :
    R,
    teste(LR).
```

Figura 17: Predicado teste

3.8 Conhecimento Perfeito

O conhecimento perfeito refere-se a toda a informação cuja totalidade é conhecida, sem qualquer grau de incerteza. Este tipo de conhecimento é constituído por factos totalmente determinados, refletindo a realidade tal como é percebida e registada pelo sistema. Na nossa Base de Conhecimento, o conhecimento perfeito está organizado em factos positivos e factos negativos, ambos explicitamente declarados.

3.8.1 Conhecimento Perfeito Positivo

O conhecimento perfeito positivo abrange todos os factos que descrevem informações confirmadas e verídicas sobre os pacientes e consultas registados. Estes factos são diretamente representados pelos predicados paciente e consulta, correspondendo a situações completamente definidas, sem qualquer dúvida ou lacuna. A informação deste tipo de conhecimento foi especificada da seguinte forma:

3.8.1.1 Factos Positivos: Pacientes

Cada paciente é representado através do predicado paciente/8, que inclui o identificador (ID), nome, data de nascimento, sexo, morada, altura, peso e histótico.

Exemplos:

```
% Pacientes Femininos (f)
paciente(1, maria, date(20, 5, 1985), f, braga, 165, 62.5, nulo_val).
paciente(2, ana, date(15, 8, 2001), f, porto, 158, 55.2, nulo_val).
paciente(3, joana, date(1, 4, 2010), f, lisboa, 140, 40.0, nulo_val).
paciente(4, luisa, date(19, 1, 1953), f, coimbra, 160, 70.3, nulo_val).
paciente(5, nadia, date(5, 12, 2005), f, faro, 150, 48.6, nulo_val).

% Pacientes Masculinos (m)
paciente(6, joao, date(3, 11, 1972), m, aveiro, 178, 85.0, nulo_val).
paciente(7, jose, date(28, 2, 1960), m, setubal, 172, 92.1, nulo_val).
paciente(8, antonio, date(10, 9, 1995), m, viseu, 185, 75.8, nulo_val).
paciente(9, manuel, date(25, 7, 1988), m, leiria, 175, 79.9, nulo_val).
paciente(10, carlos, date(17, 6, 1977), m, santarem, 180, 88.7, nulo_val).
```

Figura 18: Representação do conhecimento positivo relativamente aos pacientes

3.8.1.2 Factos Positivos: Consultas

As consultas médicas são representadas através do predicado consulta/8, que inclui o identificador da consulta, o identificador do paciente, data, a idade no momento da medição, os valores diastólico e sistólico, a pulsação registada e o estado da consulta.

Estes factos registam valores reais de tensão arterial correspondentes aos pacientes previamente definidos.

```
% Consultas Realizadas
consulta(1001, 1, date(12, 3, 2024), nulo_val, 118, 76, 72, realizada).
consulta(1002, 2, date(5, 4, 2024), nulo_val, 110, 70, 68, realizada).
consulta(1003, 3, date(22, 5, 2024), nulo_val, 102, 66, 85, realizada).
consulta(1004, 4, date(14, 6, 2024), nulo_val, 130, 84, 78, realizada).
consulta(1005, 5, date(2, 7, 2024), nulo_val, 115, 72, 74, realizada).

% Consultas Agendadas (sem medições)
consulta(1006, 6, date(19, 8, 2025), nulo_val, nulo_val, nulo_val, nulo_val, agendada).
consulta(1007, 7, date(9, 9, 2025), nulo_val, nulo_val, nulo_val, nulo_val, agendada).
consulta(1008, 8, date(1, 10, 2025), nulo_val, nulo_val, nulo_val, nulo_val, agendada).

% Consultas Canceladas (sem medições)
consulta(1009, 9, date(20, 11, 2024), nulo_val, nulo_val, nulo_val, nulo_val, cancelada).
consulta(1010, 10, date(12, 12, 2024), nulo_val, nulo_val, nulo_val, nulo_val, cancelada).
```

Figura 19: Representação do conhecimento positivo relativamente as consultas

3.8.1.3 Factos Positivos: Tensão Arterial

A classificação dos níveis de tensão arterial é representada através do predicado tensao_arterial/6, que define o seu identificador, a classificação da tensão arterial e os limites inferior e superior de valores sistólicos e diastólicos.

Estas classificações constituem o conhecimento positivo referente às faixas de tensão arterial, servindo posteriormente para a realização de diagnósticos automáticos dos pacientes.

```
% Tensão Arterial - Classificações
tensao_arterial(2001, otima,                        0, 119,   0,  79).
tensao_arterial(2002, normal,                     120, 129,  80,  84).
tensao_arterial(2003, normal_alta,                130, 139,  85,  89).
tensao_arterial(2004, hipertensao_grau1,          140, 159,  90,  99).
tensao_arterial(2005, hipertensao_grau2,          160, 179, 100, 109).
tensao_arterial(2006, hipertensao_grau3,          180, sem_limite, 110, sem_limite).
tensao_arterial(2007, hipertensao_sistolica_isolada, 140, sem_limite, 0, 89).
```

Figura 20: Representação do conhecimento positivo relativamente as classificações da tensão arterial

3.8.2 Conhecimento Negativo

O conhecimento negativo corresponde a toda a informação que foi retirada ou considerada falsa na base de conhecimento. Este tipo de conhecimento surge durante operações de involução, ou seja, quando factos que eram previamente verdadeiros deixam de o ser devido a atualizações ou eliminações. No contexto do sistema de acompanhamento da tensão arterial, o conhecimento negativo manifesta-se, por exemplo, através de:

- Remoção de um paciente da base de dados;
- Eliminação de consultas antigas ou incorretas;
- Revogação de classificações de tensão arterial que se tornaram obsoletas.

3.8.2.1 Conhecimento Negativo sobre um Paciente

A informação de que um paciente não pertence à base de conhecimento é representada como conhecimento negativo. Para tal, utiliza-se um predicado negativo, que declara explicitamente que determinado paciente não faz parte do conjunto de factos válidos do sistema.

Este predicado declara como falso que exista um paciente com essas características, reforçando o conceito de conhecimento negativo na base de dados.

```
% Pacientes que não existem ou não pertencem à clínica
-paciente(11, sofia, date(10, 3, 1999), f, braga, 162, 58.2, nulo_val).
-paciente(12, ricardo, date(7, 9, 1980), m, porto, 175, 90.1, nulo_val).
```

Figura 21: Conhecimento negativo sobre um paciente

3.8.2.2 Conhecimento Negativo sobre uma Consulta

De maneira semelhante, quando se quer indicar que um determinado evento não teve lugar, utiliza-se o conhecimento negativo. Este tipo de informação é importante para evitar a introdução de factos incorretos que violem as regras estabelecidas do sistema.

Assim, assegura-se que não serão inseridos registos inválidos, como consultas em dias em que o serviço estava inoperante, mantendo a consistência da base de conhecimento.

```
% Consultas que não aconteceram (clínica fechada, feriados, ausência médica)
-consulta(1011, 1, date(25, 12, 2024), nulo_val, nulo_val, nulo_val, nulo_val, realizada).
-consulta(1012, 3, date(1, 1, 2025), nulo_val, nulo_val, nulo_val, nulo_val, realizada).

% Consultas que não foram agendadas
-consulta(1013, 6, date(10, 7, 2025), nulo_val, nulo_val, nulo_val, nulo_val, agendada).
-consulta(1014, 9, date(18, 8, 2025), nulo_val, nulo_val, nulo_val, nulo_val, agendada).
```

Figura 22: Conhecimento negativo sobre uma consulta

3.8.2.3 Conhecimento Negativo sobre Tensão Arterial

De maneira análoga, o conhecimento negativo pode ser utilizado para indicar situações impossíveis ou inválidas dentro do domínio. Por exemplo, é declarado que não podem existir registos de tensão arterial com valores sistólicos negativos, pois tal não é fisiologicamente viável. Esta abordagem impede a inclusão de factos incorretos que contrariem as regras do sistema.

Dessa forma, garante-se que medidas inválidas não são adicionadas à base de conhecimento, mantendo a coerência e a fiabilidade dos dados clínicos.

```
% Tensões arteriais inválidas ou impossíveis
-tensao_arterial(2008, invalida, 500, 600, 400, 500).
```

Figura 23: Conhecimento negativo sobre a tensão arterial

3.9 Conhecimento Imperfeito

No contexto clínico, é comum surgirem situações em que a informação disponível sobre um paciente ou uma consulta não é completa. Essa incompletude pode manifestar-se através de dados em falta, valores desconhecidos ou informação que, por motivos específicos, não pode ser revelada. Para lidar adequadamente com estes cenários, recorre-se ao uso de valores nulos, que permitem distinguir entre conhecimento verdadeiro, falso e desconhecido, garantindo que o sistema consegue interpretar corretamente a ausência de dados.

A origem dessa falta de informação pode variar e, por isso, são considerados diferentes tipos de conhecimento imperfeito. Quando não existe qualquer valor conhecido para um determinado atributo, estamos perante conhecimento imperfeito incerto, traduzindo uma total ausência de informação. Se, pelo contrário, é conhecido apenas um conjunto limitado de valores possíveis, sem que seja possível determinar o valor exato, a situação corresponde a conhecimento imperfeito impreciso. Além destes casos, existe ainda o conhecimento imperfeito interdito, que surge quando determinada informação é intencionalmente inacessível, sendo proibida a sua consulta ou alteração. Pode também ocorrer uma forma mista, combinando partes interditas com partes simplesmente desconhecidas, refletindo situações reais em que parte da informação é confidencial e outra parte ainda não é conhecida.

Para gerir estas diversas formas de ausência ou limitação de informação, aplica-se o Pressuposto do Mundo Fechado. Segundo este princípio, tudo o que não é explicitamente declarado como verdadeiro e não é identificado como exceção é automaticamente considerado falso. Assim, a base de conhecimento consegue tratar de forma consistente factos verdadeiros, falsos e desconhecidos, sem gerar contradições.

Na secção seguinte são descritos os diferentes tipos de valores nulos utilizados, bem como a forma como cada um deles é formalmente representado no sistema.

3.9.1 Conhecimento Imperfeito Incerto

O conhecimento imperfeito incerto corresponde aos casos em que um determinado valor é totalmente desconhecido, não existe qualquer informação disponível sobre ele, nem um conjunto de possibilidades pré-definidas. Quando um parâmetro é marcado como incerto, qualquer tentativa de obter o seu valor resultará sempre em “desconhecido”. Contudo, se esse parâmetro vier a ser posteriormente atualizado com um valor concreto, passa a ser tratado como conhecimento perfeito: o novo valor será considerado verdadeiro, e todos os restantes serão avaliados como falsos, seguindo o comportamento normal do sistema.

A representação deste tipo de desconhecimento faz-se através da introdução de exceções na base de conhecimento. Estas exceções funcionam como indicadores de que há uma situação anómala, um valor que existe conceptualmente, mas cujo conteúdo é, no momento, completamente indeterminado. Esta abordagem permite que o sistema distinga claramente entre ausência de conhecimento, erro ou contradição, mantendo a coerência lógica nos restantes predicados e inferências.

Exemplo 1: O Duarte é sem-abrigo, então a sua morada é desconhecida.

```
paciente(14, duarte, date(25,11,2005), m, desconhecido, 167, 78, nulo_val).
excecao(paciente(Id, Nome, Data, Sexo, Morada, Altura, Peso, Hist)) :-
    paciente(Id, Nome, Data, Sexo, desconhecido, Altura, Peso, Hist).
```

Figura 24: Conhecimento imperfeito incerto

Exemplo 2: O paciente de Id 9 recusou-se a medir a sua pulsação.

```
consulta(2001, 9, date(20,11,2024), nulo_val, 120, 80, desconhecido, realizada).
excecao(consulta(IdC, IdP, Data, Idade, Sist, Diast, Pulso, Estado)) :-
    consulta(IdC, IdP, Data, Idade, Sist, Diast, Pulso, Estado).
```

Figura 25: Conhecimento imperfeito incerto

3.9.2 Conhecimento Imperfeito Impreciso

Quando lidamos com conhecimento imperfeito ou impreciso, existe um parâmetro cujo valor exato não conhecemos, mas sabemos que ele pertence a um determinado conjunto de valores. Caso o parâmetro não esteja dentro desse conjunto, o resultado será considerado falso.

Exemplo 1: A certidão do paciente com Id 3 foi feita alguns anos depois do seu nascimento, logo não se sabe se nasceu no ano de 2010 ou 2011.

```
excecao(paciente(3, joana, date(1,4,2010), f, lisboa, 140, 40.0, nulo_val)).
excecao(paciente(3, joana, date(1,4,2011), f, lisboa, 140, 40.0, nulo_val)).
```

Figura 26: Conhecimento imperfeito impreciso

Exemplo 2: As últimas vezes que o paciente se pesou tinha 48.6kg e 50 kg, no entanto, não sabe o seu peso atual.

```
excecao(paciente(5, nadia, date(5,12,2005), f, faro, 150, P, nulo_val)) :-
    P >= 48.6, P =< 50.
```

Figura 27: Conhecimento imperfeito impreciso

3.9.3 Conhecimento Imperfeito Interdito

Este tipo de conhecimento, além de representar um parâmetro cujo valor é incerto como “desconhecido”, impede que esse parâmetro seja atribuído a um valor específico, ou seja, ele permanece sempre restrito ou interdito.

Exemplo 1: O paciente recusa-se a dizer o seu histórico porque quer manter a sua privacidade.

```
paciente(15, ana, date(15, 8, 2001), f, porto, 158, nulo_val, nulo_val).
excecao(paciente(Id, Nome, Data, Sexo, Morada, Altura, Peso, Hist)) :-
    paciente(Id, Nome, Data, Sexo, Morada, Altura, alguem1).
interdito(alguem1).
```

Figura 28: Conhecimento imperfeito interdito

Este tipo de conhecimento requer a definição de um invariante que assegure que nenhum valor de histórico possa ser atribuído a este parâmetro.

```
+paciente(Id, Nome, Data, Sexo, Morada, Altura, Peso, Hist) :: (
    findall(H, (paciente(Id, Nome, Data, Sexo, Morada, Altura, Peso, H), interdito(H)), L),
    length(L, N),
    N == 0
).
```

Figura 29: Conhecimento imperfeito interdito

3.10 Invariantes da Inserção

Passando para os invariantes da inserção, estes foram divididos nos invariantes para a extensão de predicado Consulta e Paciente. Para os dois predicados foram efetuados invariantes para limitar:

i. Este invariante impede a inserção de uma consulta que tenha os mesmos valores de ID, paciente, data, idade, pressão sistólica e diastólica, frequência cardíaca e estado já existentes na base de conhecimento. Evita duplicação exata de consultas.

```
+consulta(ID, ID_Pac, Data, Idade, Sist, Diast, FC, Estado) ::
    (findall((ID, ID_Pac, Data, Idade, Sist, Diast, FC, Estado),
             consulta(ID, ID_Pac, Data, Idade, Sist, Diast, FC, Estado),
             S),
     length(S, N),
     N == 0).
```

Figura 30: Invariante – consulta(i)

ii. Este invariante impede a inserção de uma consulta para um paciente que não existe na base de conhecimento. Utiliza o meta-predicado si/2 para verificar a existência do paciente antes da inserção.

```
+consulta(_, ID_Pac, _, _, _, _, _, _) ::
    paciente_existe_si(ID_Pac, verdadeiro).
```

Figura 31: Invariante – consulta(ii)

iii. Este invariante garante que apenas consultas realizadas podem ter valores de pressão e frequência cardíaca preenchidos. Se a consulta estiver agendada ou cancelada, os valores devem ser nulo_val.

```
+consulta(_, _, _, _, Sist, Diast, FC, Estado) ::
    ((Estado == realizada ->
        valida_medicoes_si(Sist, Diast, FC, verdadeiro)
    ;
        Sist == nulo_val, Diast == nulo_val, FC == nulo_val)).
```

Figura 32: Invariante – consulta(iii)

iv. Este invariante garante que o estado da consulta seja válido. Os estados permitidos são agendada, realizada ou cancelada.

```
+consulta(_, _, _, _, _, _, _, Estado) ::
    valida_estado_si(Estado, verdadeiro).
```

Figura 33: Invariante – consulta(iv)

v. Este invariante impede a inserção de uma consulta cuja data seja anterior ao nascimento do paciente. Garante consistência temporal das consultas.

```
+consulta(_, ID_Pac, date(DiaC, MesC, AnoC), _, _, _, _, _) ::
    (paciente(ID_Pac, _, date(DiaN, MesN, AnoN), _, _, _, _, _),
     compara_datas_nasc(date(DiaC, MesC, AnoC), date(DiaN, MesN, AnoN))).
```

Figura 34: Invariante – consulta(v)

vi. Este invariante impede a inserção de consultas com ID já existente na base de conhecimento. Evita confusão entre consultas diferentes.

```
+consulta(ID, _, _, _, _, _, _, _) ::
    \+ consulta(ID, _, _, _, _, _, _, _).
```

Figura 35: Invariante – consulta(vi)

vii. Este invariante impede que um paciente tenha mais de uma consulta agendada ao mesmo tempo. Permite novas consultas apenas se as anteriores estiverem realizada ou cancelada.

```
+consulta(_, ID_Paciente, _Data, _, _, _, _, Estado) ::
    (findall(EstadoExistente,
             (consulta(_, ID_Paciente, _, _, _, _, _, EstadoExistente),
              EstadoExistente == agendada),
             L),
     length(L, N),
     (Estado == cancelada ; Estado == realizada ; N == 0)).
```

Figura 36: Invariante – consulta(vii)

viii. Este invariante impede que um paciente tenha duas consultas na mesma data com estado agendada ou realizada. Permite remarcar apenas se a consulta anterior estiver cancelada.

```
+consulta(_, ID_Paciente, Data, _, _, _, _, Estado) ::
    (findall(EstadoExistente,
             (consulta(_, ID_Paciente, Data, _, _, _, _, EstadoExistente),
              pertence(EstadoExistente, [agendada, realizada])),
             L),
     length(L, N),
     (Estado == cancelada ; N == 0)).
```

Figura 37: Invariante – consulta(viii)

ix. Este invariante impede a edição de consultas já realizadas. Evita que dados clínicos já validados sejam modificados.

```
+consulta(ID, ID_Paciente, _, _, _, _, _, _) ::
    (
        findall(E, consulta(ID, ID_Paciente, _, _, _, _, _, E), ListaEstados),
        (ListaEstados == []
        ; (ListaEstados = [EstadoAntigo], EstadoAntigo \= realizada))
    ).
```

Figura 38: Invariante – consulta(ix)

x. Este invariante garante que a pressão sistólica seja sempre maior que a diastólica nas consultas realizadas. Evita erros de medições clínicas.

```
+consulta(_, _, _, _, Sist, Diast, _, Estado) ::
    ((Estado == realizada -> Sist > Diast ; true)).
```

Figura 39: Invariante – consulta(x)

xi. Este invariante impede a modificação de consultas que estejam canceladas. Evita inconsistências e regressão de estados.

```
+consulta(ID, ID_Paciente, _, _, _, _, _, _Estado) ::
    (findall(EstadoAntigo,
             consulta(ID, ID_Paciente, _, _, _, _, _, EstadoAntigo),
             ListaEstados),
     (ListaEstados == [] ;
      (ListaEstados = [EstadoAntigo], EstadoAntigo \= cancelada))).
```

Figura 40: Invariante – consulta(xi)

xii. Este invariante impede a remoção de consultas realizadas. Garante que informações clínicas importantes não sejam perdidas.

```
-consulta(_, _, _, _, _, _, _, Estado) ::
    valida_remocao_si(Estado, verdadeiro).
```

Figura 41: Invariante – consulta(xii)

xiii. Este invariante impede que o estado de uma consulta seja alterado de forma a regredir no processo (por exemplo, de realizada para agendada). Permite apenas transições válidas: agendada -> cancelada, agendada -> realizada, agendada -> agendada.

```
+consulta(ID, ID_Pac, _, _, _, _, _, EstadoNovo) ::
    (findall(EstadoAntigo,
             consulta(ID, ID_Pac, _, _, _, _, _, EstadoAntigo),
             Lista),
     (Lista == [] ;
      (Lista = [EstadoAntigo],
       transicao_valida(EstadoAntigo, EstadoNovo)))).

transicao_valida(agendada, cancelada).
transicao_valida(agendada, realizada).
transicao_valida(agendada, agendada).
```

Figura 42: Invariante – consulta(xiii)

xiv. Este invariante impede que valores de pressão ou frequência cardíaca sejam inseridos em consultas agendada ou cancelada. Somente consultas realizadas podem ter medições.

```
+consulta(_, _, _, _, Sist, Diast, FC, Estado) ::
    (
        (Estado == realizada,
         valida_medicoes_si(Sist, Diast, FC, verdadeiro))
        ;
        (pertence(Estado, [agendada, cancelada]),
         Sist == nulo_val, Diast == nulo_val, FC == nulo_val)
    ).
```

Figura 43: Invariante – consulta(xiv)

xv. Este invariante garante que a idade registada na consulta corresponda à idade real do paciente na data da consulta. Evita inconsistências entre idade e data de nascimento do paciente.

```
consulta(_, ID_Pac, date(Dc, Mc, Ac), Idade, _, _, _, _) ::
    idade_corresponde_si(ID_Pac, date(Dc, Mc, Ac), Idade, verdadeiro).
```

Figura 44: Invariante – consulta(xv)

xvi. Este invariante assegura que, ao inserir um paciente, os campos fundamentais, nome, data de nascimento, altura e peso, não podem ser nulos. Esta restrição garante que não são criados registos incompletos ou inválidos, evitando conhecimento defeituoso logo na origem. Assim, apenas pacientes com dados mínimos essenciais podem entrar na base de conhecimento.

```
+paciente(_, Nome, DataNasc, _, _, Altura, Peso, _) :: (
    Nome \= nulo_val,
    DataNasc \= nulo_val,
    Altura \= nulo_val,
    Peso \= nulo_val
).
```

Figura 45: Invariante – paciente(i)

xvii. Este invariante garante a integridade referencial entre pacientes e consultas, impedindo que um paciente seja removido caso existam consultas associadas ao seu identificador. Deste modo, evita-se que fiquem registos clínicos "órfãos", protegendo a consistência do sistema e mantendo o historial médico coerente.

```
-paciente(ID, _, _, _, _, _, _, _) :: (
    findall(C_ID, consulta(C_ID, ID, _, _, _, _, _, _), S_Consultas),
    comprimento(S_Consultas, N),
    N == 0
).
```

Figura 46: Invariante – paciente(ii)

xviii. Este invariante assegura que a idade inserida numa consulta é válida. Permite apenas idades nulas (casos imperfeitos) ou valores entre 0 e 150 anos, garantindo realismo e evitando dados clinicamente impossíveis. Assim, protege-se o sistema de erros de introdução de dados e de registos inconsistentes.

```
+consulta(_, _, _, Idade, _, _, _, _) ::
    (Idade == nulo_val ; idade_valida(Idade)).
```

Figura 47: Invariante – paciente(iii)

xix. Este invariante verifica se a idade do paciente, calculada a partir do ano atual, é biologicamente plausível. Se a idade ultrapassar 150 anos, o registo é rejeitado. Este controlo evita a inserção de dados incoerentes, reforçando a qualidade e veracidade da informação clínica armazenada.

```
paciente_idade_valida(date(_, _, AnoNasc)) :-
    AnoAtual = 2025,
    Idade is AnoAtual - AnoNasc,
    idade_valida(Idade).

+paciente(_, _, DataNasc, _, _, _, _, _) :: paciente_idade_valida(DataNasc).

idade_valida(Idade) :- integer(Idade), Idade >= 0, Idade =< 150.
```

Figura 48: Invariante – paciente(iv)

xx. Este invariante garante que cada paciente tem um identificador único, impedindo a inserção de dois pacientes com o mesmo ID. A presença de duplicados iria comprometer a integridade da base de conhecimento e dificultaria o acesso correto ao histórico médico de cada pessoa. Assim, esta regra assegura unicidade e consistência estrutural dos registos.

```
+paciente(ID, Nome, DataNasc, Sexo, Morada, Altura, Peso, Historico) ::
    ( \+ paciente(ID, _, _, _, _, _, _, _) ).
```

Figura 49: Invariante – paciente(v)

3.10 Evolução e Involução da Base de Conhecimento

De forma a garantirmos que havia integridade na Base de Conhecimento, isto é, que não havia inserção de parâmetros incorretos ou impossíveis, demos uso aos invariantes que permitiram controlar a informação que era inserida. Em PROLOG, já existem predicados que nos permitem retirar e acrescentar informações, os predicados assert e retract. No entanto, estes permitem a inserção e remoção de qualquer informação sem verificar se esta é correta ou não. Assim, é crucial a criação de predicados auxiliares que garantam que a inserção de conhecimento não altera a consistência da Base de Conhecimento.

3.10.1 Evolução da Base de Conhecimento

De forma a permitir a inserção de novos elementos, criamos o predicado evolução que apenas permite a inserção caso seja válida, ou seja, vai em conta com os invariantes. A extensão deste predicado está representada de seguida:

```
evolucao(Termo) :-
    findall(Invar,+Termo::Invar,L),
    teste(L),
    insercao(Termo).

insercao(Termo) :- assert(Termo).
insercao(Termo) :- retract(Termo), !, fail.
```

Figura 50: Inserção de conhecimento executado no VSC

3.10.2 Involução da Base de Conhecimento

Quanto ao predicado involução, que limita as informações que podem ser removidas, caso a função teste funcione, é permitido o correto funcionamento da extensão do predicado involução, caso contrário não é possível a remoção desta informação, tal como, é evidente de seguida:

```
involucao(Termo) :-
    findall(Invar,-Termo::Invar,Lista),
    teste(Lista),
    retract(Termo),
    !.

remocao(Termo):-retract(Termo).
remocao(Termo):-assertz(Termo), !, fail.
```

Figura 51: Remoção de conhecimento

4 Conclusão

O presente projeto consistiu no desenvolvimento de um sistema de representação e raciocínio sobre conhecimento clínico, implementado na linguagem Prolog, que permitiu abordar de forma estruturada a gestão de conhecimento perfeito e imperfeito. A base de conhecimento concebida possibilitou o armazenamento e a manipulação de dados relativos a pacientes, consultas e medições de tensão arterial, assegurando que informações positivas, negativas ou desconhecidas fossem corretamente representadas.

Para garantir a consistência e fiabilidade da informação, foram implementados invariantes estruturais e referenciais, que impedem a inserção de dados inválidos e asseguram a coerência da base ao longo do tempo.

O sistema contemplou ainda a representação de conhecimento imperfeito, subdividido em categorias como incerto, impreciso e interdito, o que permitiu lidar de forma sistemática com lacunas e limitações presentes na informação clínica. A evolução do conhecimento foi assegurada através de predicados específicos, como evolucao e evoluirConhecimento, que possibilitam inserir, remover ou atualizar dados sem comprometer a integridade global do sistema.

Adicionalmente, foram implementados mecanismos de inferência lógica, designadamente os predicados si, siC e siD, que permitem não só responder a questões simples, mas também processar operações compostas, como conjunções e disjunções de factos, conferindo ao sistema capacidade de raciocínio sobre múltiplas condições de forma integrada e robusta. Esta abordagem reforça a fiabilidade das conclusões, mesmo perante informação parcial ou incompleta.

Em síntese, o desenvolvimento deste projeto permitiu consolidar conhecimentos fundamentais sobre programação em lógica estendida, representação de conhecimento e inferência em situações de incerteza, evidenciando a aplicabilidade de Prolog na gestão de dados clínicos e na construção de sistemas inteligentes. Tendo sido assim criado um sistema capaz de representar, evoluir e raciocinar sobre conhecimento perfeito e imperfeito, integrando mecanismos de controlo de integridade, inferência lógica e gestão de informação incompleta de forma coerente e eficaz.

5. Referências Bibliográficas

RUSSELL, S.; NORVIG, P. Artificial Intelligence: A Modern Approach. 4ª ed. Pearson, 2020. CLOCKSIN, W. F.; MELLISH, C. S. Programming in Prolog: Using the ISO Standard. 5ª ed. Springer, 2003