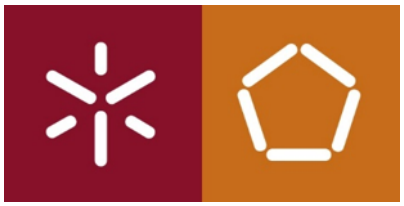

Universidade do Minho

Sistemas de Representação de Conhecimento e Raciocínio
(3º ano de curso, 2º semestre)

Exercício 1

Relatório de Desenvolvimento

Mestrado Integrado em Engenharia Informática

Realizado pelo grupo 25:
Filipa Alves dos Santos, a83631
Hugo André Coelho Cardoso, a85006
João da Cunha e Costa, a84775
Rui Alves dos Santos, a67656
Válter Ferreira Picas Carvalho, a84464

3 de Maio de 2020

Resumo

O primeiro trabalho prático realizado por este grupo no âmbito da unidade curricular *Sistemas de Representação de Conhecimento e Raciocínio* consistiu no desenvolvimento de uma base de conhecimento relativa à área dos Contratos Públicos e de um sistema de inferência ajustado à programação em lógica extendida.

No presente relatório será explicada a forma como o grupo interpretou o problema e envisionou a solução, de maneira a desenvolver uma solução robusta e capaz de acomodar todos os requisitos do enunciado, bem como alguns extras implementados para melhoria de *Quality of Service* e simplificação das operações do utilizador.

Os principais objetivos deste trabalho prático passaram por aumentar a experiência de uso da linguagem de programação simbólica *PROLOG* e do motor de inferência *SICStus*, bem como melhorar o desenho de soluções a partir de sistemas reais e a sua adaptação de forma prática e precisa. Além disso, serviu também para entender melhor o paradigma de programação em lógica extendida, a problemática de evolução e involução de conhecimento e os Pressupostos de manipulação de conhecimento - Nomes Únicos, Domínio Aberto, Mundo Aberto e Mundo Fechado.

Conteúdo

Lista de Figuras

1 Introdução

O principal objetivo deste projeto é introduzir a temática da representação dos vários tipos de conhecimento perfeito e imperfeito, assim como a sua evolução e involução para a resolução de problemas lógicos.

Como principal tema, foi pedido ao grupo para analisar o universo pertencente aos **Contratos Públicos** entre duas entidades, nomeadamente **Adjudicantes** e **Adjudicatárias**, de modo a efetuar a sua representação e evolução em **Prolog**, recorrendo ao *software* **SICStus**, que implementa este paradigma de programação simbólica.

Para conseguir representar, de forma coerente, todas as funcionalidades pretendidas pelo problema em causa, o conhecimento foi abordado de forma diferente consoante a situação que retrata. Isto é, podem existir fragmentos de conhecimento que não seja possível determinar com toda a certeza, levando à existência de valores nulos (conhecimento imperfeito) para efeitos de evolução e posterior inserção.

Para esta divisão de tratamento de dados, foi necessário incluir um sistema de inferência que agora permite dizer se um termo é verdadeiro, falso ou desconhecido; a adição de diversos invariantes que permitem a consistência da informação presente na base de conhecimento, isto é, designam restrições e regras para a remoção e inserção de conhecimento; um sistema de evolução e involução que permite alterar a base de conhecimento de forma correta e sensível, tirando partido do sistema de inferência.

2 Preliminares

O grupo realizou um trabalho de pesquisa preliminar, com o objetivo de adquirir todos os conhecimentos necessários para a realização deste trabalho prático e para a concretização dos requisitos presentes no enunciado.

Para ganhar um melhor entendimento do tema do exercício e compreender como as entidades interagiam entre si, bem como o significado dos seus parâmetros, o grupo investigou o **Código dos Contratos Públicos** e o **sistema de adjudicação** português, recorrendo, principalmente, ao portal online do governo, referenciado pelos docentes no documento de apoio fornecido.

De um ponto de vista mais técnico e de implementação, provaram-se essenciais os conhecimentos adquiridos nas aulas teórico-práticas da unidade curricular acerca do motor de inferência **PROLOG**, bem como a análise da documentação do programa **SICStus Prolog**, usado para correr a solução elaborada.

Após o trabalho de pesquisa inicial, iniciou-se então o processo de aprofundamento do estudo de **programação em lógica** na sua componente mais teórica, isto é, a maneira como este modelo de programação funciona em termos de limitações e definições básicas, que teriam de ser tidas em conta durante a elaboração do projeto.

Deste processo, o grupo concluiu que era necessário ter atenção à forma como está definida e como é manipulada a base de conhecimento, nomeadamente o **Pressuposto dos Nomes Únicos**, que designa que duas constantes diferentes - que definem valores atómicos ou objetos - dizem respeito a duas entidades diferentes; **Pressuposto do Domínio Aberto**, que indica que podem existir mais objetos para além daqueles definidos na base de conhecimento; **Pressuposto do Mundo Aberto**, que define que podem existir mais factos ou conclusões verdadeiros para além daqueles representados na base de conhecimento.

Para a **programação em lógica extendida**, como já foi mencionado, foi necessário considerar um novo valor lógico, o **desconhecido**, que indica que não é possível provar que uma determinada informação é verdadeira nem falsa; a **negação forte**, que permite definir conhecimento negativo explicitamente, indicando que o conhecimento é, de facto, falso; a **negação por falha** que, ao contrário da anterior, apenas não consegue encontrar forma de provar, usando a base de conhecimento, que a questão é verdadeira.

Relativamente ao conhecimento imperfeito, foi necessário ter em consideração que pode existir informação incompleta, definida por valores nulos, que tem de ser processada de forma diferente. Estes novos valores são:

- **Valores Incertos**: definem as situações em que determinado fragmento da informação se encontra dentro de um conjunto indeterminado de hipóteses;
- **Valores Imprecisos**: definem as situações em que parte da informação está definida num conjunto limitado de hipóteses, mas não é possível dizer com toda a certeza qual delas é;

- **Valores Interditos**: definem as situações que não é possível, efetivamente, saber qual o valor do campo da informação assim definido, por algum motivo, seja ele externo ou interno.

3 Descrição do Trabalho e Análise de Resultados

O presente capítulo está dividido em seis secções diferentes, de maneira a possibilitar a exposição organizada e bem detalhada do projeto desenvolvido.

Em primeiro lugar, é abordada a Base de Conhecimento desenvolvida, distinguindo os diferentes tipos de conhecimento existentes, o seu comportamento e significado, e demonstrando os respetivos povoamentos executados.

De seguida, entra-se em detalhe nos Invariantes desenvolvidos para o controlo de qualidade de informação e coerência da base de conhecimento, explicando como se encontram seccionados e expondo individualmente a lógica por detrás de cada invariante.

Nas duas secções posteriores, é abordada a problemática de Evolução e Involução de Conhecimento, demonstrando os predicados elaborados para a inserção e remoção de conhecimento.

A quinta secção serve como uma espécie de manual do utilizador, detalhando os predicados especificamente desenvolvidos para facilitar a inserção e remoção de conhecimento, de maneira a tornar o processo mais intuitivo e descomplicado, bem como consultar informação existente, através de identificações. Abreviadamente, são explanadas as principais Operações do Utilizador.

Por fim, são expostas as Funcionalidades Extra implementadas pelo grupo para facultar melhorias de *Quality of Service* e tornar a solução desenvolvida mais prática e útil.

Ainda no Anexo, são apresentados os Predicados Auxiliares usados no programa.

3.1 Base de Conhecimento

Foi elaborado um sistema de representação de conhecimento relativo à área de serviços contratuais e adjudicação, que é caracterizado por três principais predicados. Estas fontes de conhecimento são o **adjudicante**, a **adjudicatária** e o **contrato**.

```
% idAdjudicante,Nome,NIF,Morada -> {V,F,D}
:- dynamic adjudicante/4.

% idAdjudicataria,Nome,NIF,Morada -> {V,F,D}
:- dynamic adjudicataria/4.
```

Figura 1: Declaração e estrutura dos predicados adjudicante e adjudicatária

As entidades adjudicante e adjudicatária são caracterizadas por um identificador único e instransponível, o seu nome, número de identificação fiscal e morada.

```
% idContrato,idAdjudicante,idAdjudicataria,TipoContrato,TipoProcedimento,Descricao,Valor,Prazo,Local,Data -> {V,F,D}
:- dynamic contrato/10.
```

Figura 2: Declaração e estrutura do predicado contrato

O contrato é caracterizado por um identificador único, os identificadores das entidades adjudicante e adjudicatária entre as quais se celebra, o seu tipo, o tipo de procedimento realizado, a sua descrição, o valor acordado e o prazo, bem como o local e data de efetivação.

De maneira a tornar o sistema mais apto a lidar com situações do mundo real, em que pode existir informação rarefeita e imprecisa, o grupo estendeu o sistema à **Programação em Lógica**, de maneira a abordar também a representação de informação incompleta. Como tal, passam a existir três alternativas possíveis de resposta a questões:

- **Verdadeiro** – conclusões obtidas a partir de informação positiva;
- **Falso** – conclusões obtidas a partir de informação negativa;
- **Desconhecido** – nenhuma das conclusões anteriores é possível.

Este domínio de soluções proporciona uma maior flexibilidade no mecanismo de inferência e no processo de obtenção de conclusões, tornando o sistema mais viável para lidar com situações do mundo real, em que pode existir informação rarefeita e imprecisa.

Como forma de construir uma base de conhecimento consistente, ampla e generalizada, o grupo tomou a decisão de definir o **Pressuposto do Mundo Fechado** para os predicados estruturantes da base de conhecimento referidos acima. Como resultado disto, qualquer predicado que não esteja definido na base e não possua uma exceção é considerado conhecimento falso. Isto traz várias vantagens ao processo de inferência do programa como, por exemplo, no caso de informação imperfeita imprecisa: se não se souber o NIF exato de um

adjudicante, mas se tiver conhecimento de que está localizado num certo intervalo de valores, torna-se possível afirmar que o adjudicante em questão com qualquer NIF fora do intervalo é falso, apesar de esta informação não se encontrar explicitamente na base de conhecimento:

```
% Pressuposto do Mundo Fechado
-X :- nao(X), nao(excecao(X)).
```

Figura 3: Pressuposto do Mundo Fechado

Este tipo de raciocínio é tranversal aos predicados adjudicante, adjudicataria e contrato, dado que o PMF é declarado de forma global, como podemos observar acima.

3.1.1 Conhecimento Perfeito

Uma vez detalhadas as bases de conhecimento, o grupo procedeu ao povoamento do sistema, introduzindo factos perfeitos positivos relativos a todos os predicados mencionados anteriormente:

```
% Adjudicante: #IdAd, Nome, NIF, Morada
adjudicante(1,'Municipio de Santarem',111111111,'Amiais de Baixo, 157').
adjudicante(2,'Municipio de Barcelos',222222222,'Santa Maria, 256').
adjudicante(3,'Municipio de Tras-os-Montes',333333333,'Vale de Gouvinhas, 87').
adjudicante(4,'Municipio de Algarve',444444444,'Ribeira de Nisa, 201').
adjudicante(5,'Municipio de Estoril',555555555,'Torres Novas, 288').
adjudicante(6,'Municipio de Idanha-a-Nova',666666666,'Vila Nova da Barquinha, 302').

% Adjudicataria: #IdAda, Nome, NIF, Morada
adjudicataria(1,'Advocacia Financeira Perantes',121212121,'Amares, 223').
adjudicataria(2,'Associacao Juridica MMS',232323232,'Sequeiros, 99').
adjudicataria(3,'Combustiveis ELF',343434343,'Paredes Secas, 82').
adjudicataria(4,'Combustiveis BP',454545454,'Figueiredo, 61').
adjudicataria(5,'Junta de Advogados Pereira',565656565,'Travassos, 87').
adjudicataria(6,'Camara Municipal de Ovar',676767676,'Aguas Santas, 138').

% Contrato: #IdC, #IdAd, #IdAda, Tipo, Procedimento, Descricao, Valor, Prazo, Local, Data
contrato(1,4,2,'Aquisicao de Servicos','Consulta Previa','Assessoria Juridica',13599,547,'Alto de Basto, 124',data(2018,6,12)).
contrato(2,3,1,'Aquisicao','Ajuste Direto','Construcao de Hospital',5000,242,'Crespos, 22',data(2020,1,21)).
contrato(3,4,4,'Empreitadas','Concurso Publico','Construcao de Estradas',14900,433,'Cunhados de Ouro, 101',data(2019,9,17)).
contrato(4,6,4,'Obras Publicas','Concurso Publico','Construcao de Estradas',15000,231,'Cabeceiras de Basto, 221',data(2019,6,1)).
contrato(5,2,3,'Locacao de Bens Moveis','Ajuste Direto','Movimento Mobiliario',4900,304,'Santa Maria, 94',data(2015,2,11)).
contrato(6,3,6,'Locacao de Bens Moveis','Ajuste Direto','Movimento Mobiliario',3333,362,'Barcelos, 32',data(2020,3,7)).
contrato(7,4,1,'Empreitadas','Consulta Previa','Assessoria Juridica',14023,502,'Sobradelo da Goma, 82',data(2017,9,12)).
contrato(8,2,4,'Aquisicao de Servicos','Concurso Publico','Construcao de Estradas',20000,206,'Geraz do Minho, 24',data(2019,11,29)).
contrato(9,2,4,'Aquisicao de Servicos','Concurso Publico','Construcao de Estradas',60000,206,'Geraz do Minho, 24',data(2019,11,29)).
```

Figura 4: Base de Conhecimento Perfeito Positivo

Também para o conhecimento perfeito negativo foram introduzidos factos, recorrendo à **negação forte**. Como tal, convém salientar que é possível existir conhecimento negativo no sistema tanto por negação forte, como é o exemplo deste povoamento, como através da **negação por falha**, como no caso explicado no fim da secção 3.1.

3.1.2 Conhecimento Imperfeito

Este tipo de conhecimento diz respeito às respostas **desconhecidas** do sistema de inferência, facultadas pela extensão à Programação em Lógica, as quais podem tomar três tipos diferentes de valores nulos: **incerto**, **impreciso** e **interdito**.

```
% Adjudicante: #IdAd, Nome, NIF, Morada
-adjudicante(10,'Municipio de Amares',111222333,'Travassos, 28').
-adjudicante(11,'Municipio de Lanhoso',222333444,'Galegos, 90').
-adjudicante(12,'Municipio de Verim',333444555,'Vila Nova, 70').
-adjudicante(13,'Municipio de Esperanca',444555666,'Amares, 32').
-adjudicante(14,'Municipio de Ovar',555666000,'Ovar, 12').
-adjudicante(14,'Municipio de Ovar',555666001,'Ovar, 12').
-adjudicante(14,'Municipio de Ovar',555666002,'Ovar, 12').
-adjudicante(14,'Municipio de Ovar',555666003,'Ovar, 12').

% Adjudicataria: #IdAda, Nome, NIF, Morada
-adjudicataria(10,'Advocacia Financeira Perantes',100000000,'Sao Mamede, 12').
-adjudicataria(11,'Associacao Juridica MMS',100000001,'Souto, 99').
-adjudicataria(12,'Combustiveis ELF',100000002,'Moimenta (Santo Andre), 29').
-adjudicataria(13,'Junta de Advogados Pereira',100000003,'Vilar da Veiga, 101').

% Contrato: #IdC, #IdAd, #IdAda, Tipo, Procedimento, Descricao, Valor, Prazo, Local, Data
-contrato(21,1,3,'Obras Publicas','Consulta Previa','Construcao de Hospital',42350,523,'Sao Martinho, 24',data(2012,9,4)).
-contrato(25,2,4,'Obras Publicas','Consulta Previa','Movimento Mobiliario',12500,555,'Tamel S. Verissimo, 201',data(2013,1,23)).
-contrato(28,3,5,'Obras Publicas','Consulta Previa','Construcao de Escola',14500,209,'Vilar do Monte, 204',data(2011,2,14)).
-contrato(10,4,2,'Aquisicao de Servicos','Consulta Previa','Assessoria Juridica',13599,547,'Alvito S. Martinho, 303',data(2018,6,10)).
-contrato(11,4,2,'Aquisicao de Servicos','Ajuste Direto','Assessoria Juridica',13599,547,'Alvito S. Martinho, 303',data(2018,6,10)).
```

Figura 5: Base de Conhecimento Perfeito Negativo

Logo, da mesma maneira que a base de conhecimento foi povoada com factos correspondentes a ambos tipos de conhecimento perfeito, positivo e negativo, foi também incluído conhecimento imperfeito de todos os tipos, que será abordado de seguida.

Conhecimento Imperfeito Incerto

Este tipo de conhecimento imperfeito diz respeito a predicados em que é desconhecido um determinado parâmetro, podendo assumir um valor de um **conjunto ilimitado de hipóteses**. Dada a quantidade elevada de factos introduzidos, serão mostrados, de seguida, apenas os que dizem respeito a adjudicantes:

```
% Entidade adjudicante
% Nao se sabe a morada da entidade adjudicante 'Municipio de Sintra', cujo ID e 20 e cujo NIF e 200000000.
adjudicante(20,'Municipio de Sintra',200000000,moradaIncerta).
% Nao se sabe o NIF da entidade adjudicante 'Municipio de Beja' com ID 21 localizada na 'Rua do Bastonario, 125'.
adjudicante(21,'Municipio de Beja',nifIncerto,'Rua do Bastonario, 125').
% Nao se sabe o nome da entidade adjudicante cujo ID e 22, com o NIF 220000000 e localizada em 'Vila Nova de Gaia, 167'.
adjudicante(22,nomeIncerto,220000000,'Vila Nova de Gaia, 167').

excecao(adjudicante(IdAd,Nome,Nif,Morada)) :- adjudicante(IdAd,Nome,Nif,moradaIncerta).
excecao(adjudicante(IdAd,Nome,Nif,Morada)) :- adjudicante(IdAd,Nome,nifIncerto,Morada).
excecao(adjudicante(IdAd,Nome,Nif,Morada)) :- adjudicante(IdAd,nomeIncerto,Nif,Morada).
```

Figura 6: Parte da Base de Conhecimento Imperfeito Incerto

Como é possível observar, para representar este tipo de conhecimento é necessário declarar o predicado com um valor indefinido no parâmetro em questão, nomeadamente "nomeIncerto". Depois, é necessário declarar uma exceção que indique à base de conhecimento para caracterizar qualquer conhecimento relativo ao parâmetro do facto em questão como **desconhecido**. Isto significa que, por exemplo, a seguinte interrogação, relativa ao primeiro exemplo da imagem, daria desconhecido:

adjudicante(20,'Municipo de Sintra',200000000,'Braga').

Contudo, é de salientar que apenas interrogações que precisem valores para o parâmetro incerto têm como resposta desconhecido. Graças ao Pressuposto do Mundo Fechado, qualquer interrogação que veja outro dos campos do predicado alterado dará falso. Dito de forma mais simples, perguntar à base de conhecimento se o adjudicante 99100 vive em Braga dará desconhecido, dado que a morada é incerta. Contudo, perguntar se se chama Município de Viana dará falso, uma vez que o seu nome é conhecido.

Conhecimento Imperfeito Impreciso

O conhecimento imperfeito impreciso diz respeito a situações em que não é possível precisar o valor de um dos parâmetros do predicado, sendo que há um conjunto limitado de hipóteses. Tal como anteriormente, será exibida apenas parte do povoamento realizado, desta vez relativa às adjudicatárias:

```
% Entidade adjudicataria
% Nao se sabe se a entidade adjudicataria 'Mobiliaria Jacintos' com ID 30 e localizada em 'Braga' tenha NIF 300000003 ou 300000004
excecao(adjudicataria(30,'Mobiliaria Jacintos',300000003,'Areias de Vilar, 292')).
excecao(adjudicataria(30,'Mobiliaria Jacintos',300000004,'Areias de Vilar, 292')).

% Nao se sabe se a entidade adjudicataria com ID 31, localizada em 'Abade de Neiva, 192' e com NIF 310000000 se chama 'Mobiliaria Pinheiro'
% ou 'Mobiliaria Pinhal'
excecao(adjudicataria(31,'Mobiliaria Pinheiro',310000000,'Abade de Neiva, 192')).
excecao(adjudicataria(31,'Mobiliaria Pinhal',310000000,'Abade de Neiva, 192')).

% Nao se sabe ao certo o NIF da entidade adjudicataria 'Combustiveis McQueen', com ID 32 e localizada em 'Rota 66', mas sabe-se que
% esta entre 200000000 e 800000000
excecao(adjudicataria(32,'Combustiveis McQueen',NIF,'Rota 66')) :- NIF>=200000000, NIF=<800000000.
intervalo(adjudicataria,32,nif,(200000000,800000000)).
```

Figura 7: Parte da Base de Conhecimento Imperfeito Impreciso

Como é possível observar, há duas alternativas possíveis para a representação deste tipo de conhecimento:

- Declaração de todas as alternativas possíveis do predicado como exceções, caso os valores possíveis do parâmetro impreciso não sejam representáveis em intervalos, e quando são poucas alternativas (regra geral). Por exemplo, se o parâmetro impreciso for uma palavra (segundo exemplo da imagem) ou caso seja um número que possa assumir apenas dois valores diferentes (primeiro exemplo).
- Expressão de uma exceção do predicado com uma variável no campo do parâmetro impreciso que obedeça a um intervalo de valores (restantes exemplos da imagem).

Semelhante ao que foi descrito no tipo anterior de conhecimento, o sistema de inferência responderá **falso** se for interrogado com alternativas **diferentes** das existentes na base de conhecimento, e **desconhecido** no caso das mesmas.

Há ainda mais uma nuance no que toca aos intervalos de valores imprecisos. Por motivos que serão explicados mais à frente neste relatório, o grupo decidiu criar uma estrutura **intervalo**, que guarda o tipo de predicado com o conhecimento impreciso, o identificador do termo e o nome do parâmetro aos quais diz respeito o intervalo, bem como os limites do mesmo, num par.

```
% nomePredicado,id,nomeParametroImpreciso,(limiteInf,limiteSup)
:- dynamic intervalo/4.
```

Figura 8: Predicado intervalo e respetiva estrutura

Conhecimento Imperfeito Interdito

O terceiro e último tipo de conhecimento imperfeito corresponde a predicados que possuam um parâmetro cujo valor não seja permitido saber.

```
% E impossivel saber  a data em que o contrato com ID 50, (...) foi estabelecido.
contrato(50,1,2,'Obras Publicas','Consulta Previa','Construcao de Hospital',18500,532,'Sao Pedro do Sul, 82',dataInterdita).

excecao(contrato(IdC,IdAd,IdAda,T,Proc,Desc,V,Pra,L,D)) :- contrato(IdC,IdAd,IdAda,T,Proc,Desc,V,Pra,L,dataInterdita).

nuloInterdito(dataInterdita).

+contrato(IdC,IdAd,IdAda,T,Proc,Desc,V,Pra,L,D) ::
    (solucoes((IdC,IdAd,IdAda,T,Proc,Desc,V,Pra,L,D),
    (contrato(50,1,2,'Obras Publicas','Consulta Previa','Construcao de Hospital',18500,532,'Sao Pedro do Sul, 82',dataInterdita),
        nao(nuloInterdito(dataInterdita))), R),
    comprimento(R,0)).
```

Figura 9: Exemplo da Base de Conhecimento Imperfeito Interdito

A representação deste tipo de conhecimento é mais complexa que a dos outros dois. Repare-se no excerto do povoamento acima. Começa-se por definir o predicado, atribuindo um valor indefinido ao parâmetro em questão, "moradaInterdita", e declara-se uma exceção relativa ao parâmetro interdito, de maneira a não ser considerada conhecimento perfeito.

De seguida, instancia-se "moradaInterdita" como um valor nulo e define-se um invariante que impede a evolução de conhecimento relativo ao ajudicante em questão, com recurso à instância mencionada. A morada é impossível de saber, logo conhecimento relativo a esse adjudicante com uma morada especificada não é passível de ser adicionado à base de conhecimento.

3.2 Invariantes

Foram elaborados vários invariantes de inserção e remoção, de maneira a impor certas restrições sobre o conhecimento existente e a controlar a qualidade e consistência da informação presente na base de conhecimento.

Os invariantes foram divididos em seis grupos, de acordo com o tipo de conhecimento a que dizem respeito. Isto revelou-se necessário dado que cada tipo de conhecimento obedece a normas diferentes, e se relaciona com os outros de maneira específica e, por vezes, única.

Os seis grupos de invariantes dizem respeito os seguintes tipos de conhecimento:

- Perfeito positivo;
- Não negativo - invariantes comuns a conhecimento perfeito positivo e imperfeito, encarregam-se da validação dos parâmetros;
- Perfeito negativo;
- Imperfeito incerto ou interdito - como será observado mais à frente, estes dois tipos de conhecimento imperfeito têm comportamentos semelhantes;
- Imperfeito impreciso "**explícito**" - o grupo adotou esta notação para a primeira situação descrita em 3.1.2, dado que o utilizador declara **explicitamente** todas as alternativas possíveis do conhecimento em questão;
- Imperfeito impreciso "**implícito**" - diz respeito ao uso de intervalos de valores para declarar parâmetros desconhecidos, uma vez que é **implícito** que qualquer valor do intervalo é passível de ser o verdadeiro.

De maneira a diferenciar os invariantes, estes possuem dois argumentos - o termo que se pretende inserir e o seu tipo de conhecimento - um dos supramencionados.

3.2.1 Predicados Auxiliares

Antes de mais, serão explicados alguns predicados auxiliares que são utilizados com grande frequência nos invariantes, com o intuito de simplificar a sua sintaxe e evitar a repetição pontual de código.

Os dois predicados seguintes verificam se o número de adjudicantes na base de conhecimento com o identificador especificado é igual a N, no primeiro caso, ou o número de adjudicantes com esse id e de exceções dos mesmos, no segundo caso. Esta distinção é necessária, como será demonstrado mais à frente. Existem predicados análogos a estes dois para adjudicatárias e contratos.

De seguida, o predicado **solucoesIntervaloId** verifica se existe alguma exceção na base de conhecimento relativo a um termo cujo valor de certo parâmetro pertença a um intervalo, sem ser possível apurar o valor concreto.

```
% Ve se o numero existente de adjudicantes com o id dado e N
solucoesAdjudicanteId(IdAd,N) :-
    solucoes(IdAd, adjudicante(IdAd,_,_,_), R),
    comprimento(R,N).

% Ve se o numero existente de adjudicantes e excecoes dos mesmos com o id dado e N
solucoesAdjudicanteExcecaoId(IdAd,N) :-
    solucoes(IdAd, (adjudicante(IdAd,_,_,_); excecao(adjudicante(IdAd,_,_,_))), R),
    comprimento(R,N).
```

Figura 10: Predicados auxiliares **solucoesAdjudicante** e **solucoesAdjudicanteExcecao**

```
% Verifica que nao ha conhecimento impreciso implicito com o mesmo id
solucoesIntervaloId(Predicado,Id) :- nao(intervaloImprecisoId(Predicado,Id,_)).
```

Figura 11: Predicado auxiliar **solucoesIntervaloId**

O **solucoesSinalContrario** garante que o termo que se pretende inserir não existe já com o sinal oposto no sistema. Por exemplo, caso tentemos inserir um termo perfeito positivo, não pode existir já um termo negativo equivalente, uma vez que isso seria contraditório.

```
% Nao pode existir um termo igual ao novo com sinal oposto
solucoesSinalContrario(X) :- solucoes(X,-X,R), comprimento(R,0).
```

Figura 12: Predicado auxiliar **solucoesSinalContrario**

O predicado **solucoesExcecao** verifica se existem N exceções do termo especificado.

```
% Nao pode existir uma excecao do termo que se pretende inserir
solucoesExcecao(X,N) :- solucoes(excecao(X),excecao(X),R), comprimento(R,N).
```

Figura 13: Predicado auxiliar **solucoesExcecao**

3.2.2 Conhecimento Positivo

Em primeiro lugar, foi realizado em estudo das interações entre o conhecimento perfeito positivo e os outros tipos de conhecimento. Foram identificadas as seguintes restrições à gestão de conhecimento positivo, em relação aos seguintes:

- **Perfeito Positivo**: os identificadores dos termos são únicos, dentro do seu predicado, logo não devem existir adjudicantes, adjudicatárias ou contratos com identificadores repetidos;

- **Perfeito Negativo**: não podem existir termos positivos e negativos iguais, uma vez que se contradiriam. Contudo, podem existir termos com o mesmo identificador, desde que não sejam totalmente iguais – se um adjudicante está localizado em A, pode-se dizer que é falso que está localizado em B;
- **Imperfeito Incerto/Interdito**: não podem ter o mesmo identificador. Conhecimento incerto indica que não se sabe um dado parâmetro e um termo positivo com o mesmo identificador significaria que afinal se sabe, o que é contraditório;
- **Imperfeito Impreciso**: não podem ter o mesmo identificador. Se existe conhecimento impreciso a indicar que não se sabe se um parâmetro é X ou Y, não pode existir um termo positivo a dizer que afinal se sabe e é, de facto, X/Y, ou que na verdade não é nenhum deles (diferente de X/Y).

Como tal, foram elaborados invariantes com estas limitações em mente. O primeiro invariante garante que não é possível inserir conhecimento positivo que já se encontre expresso na base de conhecimento sob a forma de conhecimento negativo, o que seria contraditório.

```
% Informacao positiva nao pode contrariar informacao negativa existente
+(X,positivo) :: solucoesSinalContrario(X).
```

Figura 14: Primeiro invariante geral de conhecimento positivo

É também possível observar acima a estrutura dos dois argumentos referida no fim de 3.2. O seguinte invariante assegura que não é possível inserir conhecimento perfeito positivo equivalente a exceções presentes na base de conhecimento - ou seja, conhecimento imperfeito **explícito**:

```
% Informacao positiva nao pode contrariar excecoes
+(X,positivo) :: solucoesExcecao(X,0).
```

Figura 15: Segundo invariante geral de conhecimento positivo

Quanto aos outros tipos de conhecimento imperfeito, não foi necessário elaborar um invariante especificamente para cada um desses casos, dado que já estão englobados nos restantes invariantes de conhecimento perfeito positivo que serão abordados de seguida. Foram desenvolvidos invariantes específicos a cada predicado de conhecimento armazenado.

Aquando da inserção de um novo termo, há vários requerimentos que é necessário verificar. Tenha-se em conta o caso dos adjudicantes. Em primeiro lugar, não podem existir adjudicantes com identificadores iguais na base de conhecimento, dado que o identificador é único e instranponível, dizendo respeito a um e um só adjudicante.

Isto inclui adjudicantes perfeitos e imperfeitos: se existir um predicado que indica que a morada de um certo adjudicante é interdita, não faz sentido inserir um termo perfeito positivo relativo ao mesmo.

```
% Id do adjudicante unico e intransponivel
+(adjudicante(IdAd,_,_,_),positivo) ::
    (solucoesIntervaloId(adjudicante,IdAd), solucoesAdjudicanteExcecaoId(IdAd,1)).

% Id da adjudicataria unico e intransponivel
+(adjudicataria(IdAda,_,_,_),positivo) ::
    (solucoesIntervaloId(adjudicataria,IdAda), solucoesAdjudicatariaExcecaoId(IdAda,1)).

% Id do contrato unico e intransponivel
+(contrato(IdC,_,_,_,_,_,_,_,_,_),positivo) ::
    (solucoesIntervaloId(contrato,IdC), solucoesContratoExcecaoId(IdC,1)).
```

Figura 16: Restantes invariantes de inserção de conhecimento positivo

Assim, os invariantes começam por verificar que não há nenhum termo **impreciso implícito**, assegurando a inexistência de um predicado **intervalo** com o identificador da entidade em questão. De seguida, averigua se existe apenas um predicado com o id dado na base de conhecimento. Isto cobre não só os casos de conhecimento **perfeito positivo**, como de **imperfeito incerto** e **interdito**, uma vez que, como foi explicado em 3.1.2, a inserção desses dois tipos de conhecimento implica a introdução não só de uma exceção, como também de um adjudicante (ou outro predicado semelhante).

Quanto à remoção de conhecimento positivo, os contratos possuem uma regra diferente dos restantes predicados. O grupo estabeleceu que apenas deve ser possível retirar ajudicantes e adjudicatárias da base de conhecimento que não tenham nenhum contrato associado, de maneira a não deixar lacunas na consulta de informação acerca de contratos.

```
% Nao e permitido remover adjudicantes com contratos associados
-(adjudicante(IdAd,_,_,_),positivo) ::
    (solucoes(IdAd, contrato(_,IdAd,_,_,_,_,_,_,_,_), R),
    comprimento(R,0)).

% Nao e permitido remover adjudicatarias com contratos associados
-(adjudicataria(IdAda,_,_,_),positivo) ::
    (solucoes(IdAda, contrato(_,_,IdAda,_,_,_,_,_,_,_), R),
    comprimento(R,0)).
```

Figura 17: Invariantes de remoção de adjudicantes e adjudicatárias

Quanto aos contratos, o grupo estabeleceu que não deve ser possível remover contratos ainda vigentes da base de conhecimento, ou seja, cujo prazo ainda não tenha expirado até ao dia de hoje.

```
% Não remover contratos vigentes
-(contrato(_,_,_,_,_,_,_,_,P,_,D),positivo) :: prazoExpirado(D,P).
```

Figura 18: Invariante de remoção de contratos

3.2.3 Conhecimento Não Negativo

Os invariantes apresentados nesta secção dizem respeito não só a conhecimento perfeito positivo, como a imperfeito incerto e interdito. O seu propósito é verificar que os parâmetros dos predicados inseridos na base de conhecimento são todos válidos, o que deve ser verdade para qualquer tipo de conhecimento que insira um adjudicante, adjudicatária ou contrato.

Estas verificações também são executadas na inserção de exceções, no caso de conhecimento imperfeito, mas como a manobragem de exceções é diferente da dos predicados isolados, não está incluída aqui.

Para distinção, pode-se dizer que os invariantes de conhecimento positivo encarregam-se da coerência e consistência da informação na base de conhecimento, verificando que não há contradições, enquanto que os invariantes de conhecimento não negativo asseguram que a informação faz sentido, no que toca a questões mais relacionadas com o contexto do problema.

Adjudicantes e Adjudicatárias

Estes predicados possuem um NIF, que deve ser único e pessoal. Além disso, deve ter 9 dígitos para ser um NIF válido (assegurado pelo predicado **nifValido**), não podendo o primeiro dígito ser um 0. Contudo, para estas verificações fazerem sentido, é preciso garantir que não estamos perante um caso de conhecimento imperfeito, com um NIF interdito ou incerto. Esta primeira condição é necessária, pois sem ela estes casos de conhecimento imperfeito nunca satisfariam os restantes requisitos, acabando por ser sempre descartados.

```
% NIF valido, unico e intransponivel
+(adjudicante(_,_,NIF,_),naoNegativo) ::
    (nifImperfeito(NIF);
    (nifValido(NIF), solucoes(NIF, adjudicante(_,_,NIF,_), R), comprimento(R,1))).

% NIF valido, unico e intransponivel
+(adjudicataria(_,_,NIF,_),naoNegativo) ::
    (nifImperfeito(NIF);
    (nifValido(NIF), solucoes(NIF, adjudicataria(_,_,NIF,_), R), comprimento(R,1))).
```

Figura 19: Invariantes de verificação do NIF

Contratos

De modo a apurar a validade de um contrato, as seguintes condições devem ser respeitadas:

- Os identificadores do adjudicante e da adjudicatária entre os quais se celebrou o contrato devem ser válidos e dizer respeito a entidades presentes na base;
- O valor e prazo devem ser números inteiros não negativos;
- O tipo de procedimento deve estar incluído no conjunto de hipóteses indicado pelos docentes e a data deve também dizer respeito a uma data real.

Os invariantes respetivos implementam a mesma verificação de conhecimento imperfeito que é realizada com o NIF nas outras entidades, de maneira a possibilitar a inserção tanto de conhecimento perfeito como imperfeito.

```
% Id do adjudicante associado valido
+(contrato(_,IdAd,_,_,_,_,_,_,_,_),naoNegativo) ::
    (idAdImperfeito(IdAd); solucoesAdjudicanteId(IdAd,1)).

% Id da adjudicataria associado valido
+(contrato(_,_,IdAda,_,_,_,_,_,_,_),naoNegativo) ::
    (idAdaImperfeito(IdAda); solucoesAdjudicatariaId(IdAda,1)).

% Valor valido
+(contrato(_,_,_,_,_,_,_,V,_,_),naoNegativo) ::
    (valorImperfeito(V); numeroValido(V)).

% Prazo valido
+(contrato(_,_,_,_,_,_,_,_,P,_),naoNegativo) ::
    (prazoImperfeito(P); numeroValido(P)).

% Tipo de procedimento valido
+(contrato(_,_,_,_,TP,_,_,_,_,_),naoNegativo) ::
    (procedimentoImperfeito(TP); tipoProcedimento(TP)).

% Data valida
+(contrato(_,_,_,_,_,_,_,_,_,D),naoNegativo) ::
    (dataImperfeita(D); dataValida(D)).
```

Figura 20: Invariantes de verificação dos parâmetros do contrato

Os restantes invariantes são mais detalhados e entram no domínio da adjudicação propriamente dita, enquanto que até agora as verificações feitas eram mais por uma questão de lógica.

Tal como é referido no documento de apoio ao trabalho fornecido pelos docentes, há condições obrigatórias para qualquer contrato por ajuste direto (tipo de procedimento): o valor deve ser igual ou inferior a 5000 euros, deve tratar-se de um contrato de aquisição, locação de bems móveis ou aquisição de serviços (tipo de contrato) e deve ter um prazo de vigência até 1 ano, inclusivé, a contar da decisão de adjudicação.

```
% Condicoes de contrato por ajuste direto
+(contrato(_,_,_,TC,'Ajuste Direto',_,V,P,_,_),naoNegativo) ::
    ((tipoImperfeito(TC); valorImperfeito(V); prazoImperfeito(P));
    (V =< 5000, P =< 365, tipoContratoAjusteDireto(TC))).
```

Figura 21: Invariante de conhecimento não negativo relativo a contratos

Além disso, todos os contratos devem obedecer a uma regra que dita que uma entidade adjudicante não pode convidar a mesma empresa para celebrar um contrato com prestações de serviço do mesmo tipo ou idênticas às dos contratos que já lhe foram atribuídos, no ano económico em curso ou nos dois anos económicos anteriores, sempre que o preço contratual acumulado dos contratos já celebrados (não incluindo o contrato que se pretende celebrar) seja igual ou superior a 75.000 euros.

```
% Regra dos 3 anos valida para todos os contratos
+(contrato(IdC,IdAd,IdAda,TC,_,_,Valor,_,_,D),naoNegativo) ::
    ((idAdImperfeito(IdAd); idAdaImperfeito(IdAda); tipoImperfeito(TC); valorImperfeito(Valor); dataImperfeita(D));
    (elementoData(D,ano,Ano), DoisAnosAtras is Ano-2,
    solucoes(V, (contrato(Id,IdAd,IdAda,TC,_,_,V,_,_,data(A,_,_)),
        Id =\= IdC, A =< Ano, A >= DoisAnosAtras), R),
    valorAcumulado(R,Total), Total < 75000)).
```

Figura 22: Invariante de conhecimento não negativo relativo a contratos

3.2.4 Conhecimento Negativo

Semelhante ao estudo realizado para o conhecimento positivo, foram também identificadas as restrições do conhecimento negativo em relação aos outros tipos de conhecimento:

- **Perfeito Positivo**: mesma lógica que a interação positivo-negativo, não pode haver termos completamente iguais, mas podem ter o mesmo id desde que algum dos outros parâmetros divirja;
- **Perfeito Negativo**: conclusão igual à de cima, não pode haver termos completamente iguais, mas podem ter o mesmo id desde que algum dos parâmetros divirja – faz sentido dizer, por exemplo, que o valor de um contrato não é 5000 (um termo negativo) nem 7000 (outro termo negativo). Em teoria, é possível ter infinitos predicados negativos em relação ao mesmo termo, dado que, sabendo a informação correta do mesmo (predicado positivo), todas as outras possibilidades são falsas;
- **Imperfeito Incerto/Interdito**: não há restrições – mesmo não sabendo ao certo o valor de um parâmetro, é possível saber que, pelo menos, não é X;
- **Imperfeito Impreciso**: não há restrições no que toca a intervalos, porém não pode haver conhecimento negativo igual a conhecimento impreciso implícito (exceções explicitadas na sua totalidade).

Relativamente ao conhecimento negativo, o primeiro invariante garante que não existe já um termo positivo igual ao novo termo negativo, de maneira a manter a informação coerente.

```
% Informacao negativa nao pode contrariar informacao positiva existente
+(-X,negativo) :: (solucoes(-X,X,R), comprimento(R,0)).
```

Figura 23: Primeiro invariante de conhecimento negativo

Os invariantes seguintes garantem que não existe conhecimento negativo repetido, isto é, termos totalmente iguais.

```
% Nao permitir adjudicantes negativos repetidos
+(-adjudicante(IdAd,_,_,_),negativo) ::
    (adjudicanteNegId(IdAd,R), diferentes(R,D), comprimento(R,N), !, D=:=N).

% Nao permitir adjudicatarias negativas repetidas
+(-adjudicataria(IdAda,_,_,_),negativo) ::
    (adjudicatariaNegId(IdAda,R), diferentes(R,D), comprimento(R,N), !, D=:=N).

% Nao permitir contratos negativos repetidos
+(-contrato(IdC,_,_,_,_,_,_,_,_,_),negativo) ::
    (contratoNegId(IdC,R), diferentes(R,D), comprimento(R,N), !, D=:=N).
```

Figura 24: Invariante para controlar conhecimento negativo repetido

Por fim, criaram-se os restantes invariantes para verificar que o novo termo negativo não é igual a nenhuma exceção **explícita** da base de conhecimento. Para este efeito, uma de duas condições deve verificar-se: existir um **intervalo** imperfeito relativo ao mesmo termo (a função **solucoesExcecao** verifica se **não** existe um intervalo), dado que os dois tipos de conhecimento impreciso são mutuamente exclusivos – neste caso, há liberdade total para inserir termos negativos -, ou não existir nenhuma exceção igual ao novo termo (impreciso implícito).

```
% Nao inserir adjudicante negativo igual a conhecimento imperfeito ja existente
+(-adjudicante(IdAd,N,NIF,M),negativo) ::
    (nao(solucoesIntervaloId(adjudicante,IdAd));
    (solucoesExcecao(adjudicante(IdAd,N,NIF,M),0);
    adjudicanteId(IdAd,adjudicante(_,P2,P3,P4)), nao(adjudiPerfeito(P2,P3,P4)))).

% Nao inserir adjudicataria negativa igual a conhecimento imperfeito ja existente
+(-adjudicataria(IdAda,N,NIF,M),negativo) ::
    (nao(solucoesIntervaloId(adjudicataria,IdAda));
    (solucoesExcecao(adjudicataria(IdAda,N,NIF,M),0);
    adjudicatariaId(IdAda,adjudicataria(_,P2,P3,P4)), nao(adjudiPerfeito(P2,P3,P4)))).

% Nao inserir contrato negativo igual a conhecimento imperfeito ja existente
+(-contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,D),negativo) ::
    (nao(solucoesIntervaloId(contrato,IdC));
    (solucoesExcecao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,D),0);
    contrato(IdC,contrato(_,P2,P3,P4,P5,P6,P7,P8,P9,P10)), nao(contratoPerfeito(P2,P3,P4,P5,P6,P7,P8,P9,P10)))).
```

Figura 25: Últimos invariantes de conhecimento negativo

Esta distinção é necessária porque, caso exista um intervalo de valores imprecisos, qualquer termo com um desses valores é considerado uma exceção, o que impediria a inserção de um termo negativo igual, o que não é o objetivo. Com esta disjunção, torna-se possível exercer as regras de ambos os tipos de conhecimento impreciso.

3.2.5 Conhecimento Imperfeito Incerto e Interdito

Estes dois tipos de conhecimento imperfeito são muito semelhantes: em ambos é desconhecida uma certa característica de um predicado, seja por não ser possível especificar o seu valor de entre um conjunto infinito de possibilidades (incerto) ou por ser impossível descobrir (interdito).

Como tal, as normas para a sua gestão numa base de conhecimento são iguais. As suas relações com os outros tipos de conhecimento são explicadas de seguida:

- **Perfeito Positivo**: não podem ter termos com o mesmo identificador, pois um termo positivo constitui um facto verdadeiro e completo, enquanto que conhecimento imperfeito nunca permite conhecer todos os parâmetros de um termo com certeza;
- **Perfeito Negativo**: não há restrições. É possível não se saber a morada de um adjudicante, mas saber-se que não é 'Braga';
- **Imperfeito Incerto/Interdito**: não podem ter o mesmo identificador. Se são sabidos todos os parâmetros de um contrato exceto o valor, que não é possível especificar de entre um conjunto indeterminado de valores, não faz sentido afirmar que um dos parâmetros conhecidos também é incerto ou interdito, nem mesmo o valor, dado que isso seria repetir informação (conhecimento incerto) ou contrariar (interdito);
- **Imperfeito Impreciso**: pelo motivo anterior, também não podem ter o mesmo id. Diferentes tipos de conhecimento imperfeito são mutuamente exclusivos.

Assim, o grupo começou por elaborar invariantes que impossibilitam a existência de ids repetidos, no caso de conhecimento positivo, incerto ou interdito:

```
% Id do adjudicante unico e intransponivel
+(adjudicante(IdAd,_,_,_),incertoInterdito) ::
    (solucoesIntervaloId(adjudicante,IdAd), solucoesAdjudicanteId(IdAd,1)).

% Id da adjudicataria unico e intransponivel
+(adjudicataria(IdAda,_,_,_),incertoInterdito) ::
    (solucoesIntervaloId(adjudicataria,IdAda), solucoesAdjudicatariaId(IdAda,1)).

% Id do contrato unico e intransponivel
+(contrato(IdC,_,_,_,_,_,_,_,_,_),incertoInterdito) ::
    (solucoesIntervaloId(contrato,IdC), solucoesContratoId(IdC,1)).
```

Figura 26: Primeiros invariantes de conhecimento incerto/interdito

Por fim, foram criados invariantes para garantir que não é possível inserir ids repetidos de conhecimento impreciso.

```
% Nao podem existir excecoes com o id do novo termo
+(adjudicante(IdAd,N,NIF,M),incertoInterdito) ::
    (solucoes(IdAd, excecao(adjudicante(IdAd,_,_,_)), R), comprimento(R,Num),
    ((existeExcecaoAdjudi(IdAd,N,NIF,M), Num =:= 1); Num =:= 0)).

% Nao podem existir excecoes com o id do novo termo
+(adjudicataria(IdAda,N,NIF,M),incertoInterdito) ::
    (solucoes(IdAda, excecao(adjudicataria(IdAda,_,_,_)), R), comprimento(R,Num),
    ((existeExcecaoAdjudi(IdAda,N,NIF,M), Num =:= 1); Num =:= 0)).

% Nao podem existir excecoes com o id do novo termo
+(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,D),incertoInterdito) ::
    (solucoes(IdC, excecao(contrato(IdC,_,_,_,_,_,_,_,_,_)), R), comprimento(R,Num),
    ((existeExcecaoContrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,D), Num =:= 1); Num =:= 0)).
```

Figura 27: Restantes invariantes de conhecimento incerto/interdito

Se a exceção relativa ao novo termo que é necessário inserir já existir na base de conhecimento, o novo termo inserido já será considerado desconhecido pelo sistema de inferência, portanto, nesse caso, a lista de exceções com tal id deve ter um único elemento. Caso contrário, ainda é considerado falso e não desconhecido, logo a lista de exceções deve ser vazia.

3.2.6 Conhecimento Imperfeito Impreciso Explícito

Este tipo de conhecimento é representado através de exceções relativas a termos "completos- com todos os parâmetros bem definidos - que representam hipóteses dos valores que

o termo pode tomar, não sendo possível, no entanto, dizer com certezas qual dos termos em questão está correto.

O grupo teve de averiguar que interações este tipo de conhecimento podia ter com os demais:

- **Perfeito Positivo**: não podem existir termos do mesmo predicado com identificadores iguais - conhecimento perfeito e imperfeito são mutuamente exclusivos, uma vez que as suas definições se contrariam;
- **Perfeito Negativo**: os dois tipos de conhecimento podem coexistir, em relação a um predicado com um dado identificador - saber que o NIF de um adjudicante é 111111111 ou 333333333 não invalida o facto de que é falso que o NIF é 222222222. Não pode, contudo, existir um termo negativo e um impreciso completamente iguais, pois isso significaria que já se sabe que umas das alternativas é falsa, pelo que a sua inserção é redundante;
- **Imperfeito Incerto/Interdito**: não podem ter o mesmo id, pois os vários tipos de conhecimento imperfeito são mutuamente exclusivos;
- **Imperfeito Impreciso**: não podem existir alternativas repetidas, e conhecimento impreciso implícito e explícito não podem coexistir em relação ao mesmo termo, pois isso implicaria informação repetida ou contraditória.

Desta maneira, foram desenvolvidos os seguintes invariantes para controlar a inserção deste tipo de conhecimento. Em primeiro lugar, criaram-se invariantes para garantir que não existe conhecimento impreciso implícito (intervalo de valores) ou perfeito positivo, imperfeito incerto ou imperfeito interdito (termo com o mesmo id) relativamente ao predicado a que diz respeito a nova exceção que se está a tentar inserir.

```
% Id do adjudicante unico e intransponivel
+(excecao(adjudicante(IdAd,_,_,_)),imprecisoExplicito) ::
    (solucoesIntervaloId(adjudicante,IdAd), solucoesAdjudicanteId(IdAd,1)).

% Id da adjudicataria unico e intransponivel
+(excecao(adjudicataria(IdAd,_,_,_)),imprecisoExplicito) ::
    (solucoesIntervaloId(adjudicataria,IdAda), solucoesAdjudicatariaId(IdAda,1)).

% Id do contrato unico e intransponivel
+(excecao(contrato(IdC,_,_,_,_,_,_,_,_,_,_)),imprecisoExplicito) ::
    (solucoesIntervaloId(contrato,dC), solucoesContratoId(IdC,1)).
```

Figura 28: Primeiros invariantes de conhecimento impreciso explícito

De seguida, verifica-se que não existe um termo negativo igual àquele cuja exceção se pretende inserir.

```
% Nao pode ser igual a um termo negativo existente
+(excecao(X),imprecisoExplicito) :: solucoesSinalContrario(X).
```

Figura 29: Invariante de conhecimento impreciso explícito relativo a termos negativos

Por fim, criou-se o seguinte invariante para garantir que não se está a inserir uma exceção repetida na base de conhecimento.

```
% Nao ha excecoes repetidas
+(excecao(X),imprecisoExplicito) :: solucoesExcecao(X,1).
```

Figura 30: Último invariante de conhecimento impreciso explícito

3.2.7 Conhecimento Imperfeito Impreciso Implícito

Este tipo de conhecimento impreciso retrata-se com intervalos onde se encontra o verdadeiro valor do parâmetro de um dado predicado, sem ser possível precisar qual.

Uma vez que também é um tipo de conhecimento imperfeito impreciso, a maneira como interage com os outros tipos de conhecimento é igual à apresentada em 3.2.6, com uma diferença: um intervalo de valores imprecisos pode conter um valor falso - saber que o valor de um contrato não é 5000 não invalida o facto de se encontrar entre 4000 e 6000.

Assim, os primeiros invariantes desenvolvidos servem para garantir que a nova exceção não contradiz nenhum conhecimento já existente, do seu próprio tipo (intervalo de valores) ou perfeito positivo, imperfeito incerto ou imperfeito interdito (termo com o mesmo id), bem como impreciso explícito (exceções com o mesmo id).

```
% Id do adjudicante unico e intransponivel
+(excecao(adjudicante(IdAd,_,_,_)),imprecisoImplicito) ::
    (solucoesIntervaloId(adjudicante,IdAd), solucoesAdjudicanteExcecaoId(IdAd,0)).

% Id da adjudicataria unico e intransponivel
+(excecao(adjudicataria(IdAda,_,_,_)),imprecisoImplicito) ::
    (solucoesIntervaloId(adjudicataria,IdAda), solucoesAdjudicatariaExcecaoId(IdAda,0)).

% Id do contrato unico e intransponivel
+(excecao(contrato(IdC,_,_,_,_,_,_,_,_,_)),imprecisoImplicito) ::
    (solucoesIntervaloId(contrato,IdC), solucoesContratoExcecaoId(IdC,0)).
```

Figura 31: Primeiros invariantes de conhecimento impreciso implícito

Os restantes invariantes desenvolvidos têm como propósito impedir a inserção de conhecimento impreciso com intervalos de valores totalmente falsos.

Considere-se, por exemplo, um dado contrato cujo prazo se encontra entre 200 e 202 dias.

Se esses três termos já estiverem declarados como conhecimento falso, não deve ser possível inserir o intervalo mencionado, pois seria contraditório.

Existe um destes invariantes para cada cenário de informação imprecisa por intervalos possível na base de conhecimento, mas seguem todos a mesma linha de raciocínio. Por este motivo, serão apenas exibidos alguns exemplos deste código:

```
% Adjudicante com intervalo valido de valores para o NIF
+(excecao(adjudicante(IdAd,N,(Inf,Sup),M)),imprecisoImplicito) ::
    intervaloNaoTodoNegativo(adjudicante(IdAd,N,(Inf,Sup),M)).

% Adjudicataria com intervalo valido de valores para o NIF
+(excecao(adjudicataria(IdAda,N,(Inf,Sup),M)),imprecisoImplicito) ::
    intervaloNaoTodoNegativo(adjudicataria(IdAda,N,(Inf,Sup),M)).

% Contrato com intervalo valido de valores para o valor
+(excecao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,(Inf,Sup),P,L,D)),imprecisoImplicito) ::
    intervaloNaoTodoNegativo(contrato(IdC,IdAd,IdAda,TC,TP,Desc,(Inf,Sup),P,L,D)).
```

Figura 32: Restantes invariantes de conhecimento impreciso implícito

O predicado **intervaloNaoTodoNegativo** pode ser consultado no anexo, para uma compreensão mais profunda de como o mesmo verifica a condição descrita acima.

3.3 Evolução de Conhecimento

Tendo em conta o panorama da Extensão à Programação em Lógica, o objetivo do grupo consistiu em possibilitar a inserção e remoção de qualquer tipo de conhecimento, de maneira a tornar a base de conhecimento mais completa e diversificada, embora mantendo consistência na sua informação. Isto traduz-se na gestão não só de factos verdadeiros e falsos (conhecimento perfeito), como de termos incompletos, em que não seja possível apurar, com certeza, um dos seus parâmetros (conhecimento imperfeito).

Para lidar com a problemática da evolução de conhecimento, foi necessário desenvolver procedimentos específicos a cada tipo de conhecimento, dado que se regem por normas e formatos diferentes.

3.3.1 Conhecimento Perfeito

Em primeiro lugar, foram desenvolvidos predicados de evolução que permitem a inserção de novo conhecimento. Estes predicados reúnem todos os invariantes aplicáveis ao termo novo e inserem o mesmo na base de conhecimento, testando, de seguida, se obedece aos invariantes mencionados. Caso isto se verifique, o novo termo fica assim introduzido na base, caso contrário é removido.

Para a inserção de conhecimento positivo, são testados os invariantes de conhecimento positivo e de não negativo:

```
% Inserir conhecimento perfeito positivo na base de conhecimento
evolucaoPos(Termo) :-
    solucoes(Invariante, (+(Termo,positivo)::Invariante; +(Termo,naoNegativo)::Invariante), Lista),
    insercao(Termo),
    teste(Lista).
```

Figura 33: Predicado de evolução de conhecimento positivo

Já para o conhecimento negativo, como é possível observar na imagem seguinte, a sua evolução resulta no armazenamento do termo precedido de um sinal negativo, enquanto que no caso do conhecimento positivo, o sinal é omitido. O termo novo é testado com os invariantes de conhecimento negativo, obviamente:

```
% Inserir conhecimento perfeito negativo na base de conhecimento
evolucaoNeg(Termo) :-
    solucoes(Invariante, +(-Termo,negativo)::Invariante, Lista),
    insercao(-Termo),
    teste(Lista).
```

Figura 34: Predicado de evolução de conhecimento negativo

3.3.2 Conhecimento Imperfeito Incerto

A evolução de conhecimento imperfeito incerto é um processo mais complexo do que as evoluções que vimos anteriormente, dado que a sua escrita envolve mais do que a declaração do termo, como é o caso do conhecimento perfeito. É necessário também criar uma exceção, de maneira a indicar ao sistema de inferência que o parâmetro em questão do termo é desconhecido, como foi abordado no capitulo 3.1.2.

Para começar, foi criado o predicado de evolução **evolucaoIncertoInterdito** que filtra os invariantes de conhecimento incerto/interdito e não negativo, insere o termo na base de conhecimento e procede ao teste do mesmo, removendo-o de novo, se necessário.

```
evolucaoIncertoInterdito(Termo) :-
    solucoes(Invariante, (+(Termo,incertoInterdito)::Invariante; +(Termo,naoNegativo)::Invariante), Lista),
    insercao(Termo),
    teste(Lista).
```

Figura 35: Predicado de evolução de conhecimento incerto/interdito

Foi necessário desenvolver predicados de evolução para todos os casos possíveis de conhecimento incerto: para cada fonte de conhecimento, um por cada parâmetro passível de ser desconhecido. Os predicados de evolução são distuinguíveis pelo termo incerto, sendo que o sistema de inferência faz uso do predicado que pretende por **pattern matching**.

O grupo definiu que qualquer parâmetro exceto o identificador de um termo podia ser incerto. Não faz sentido considerar o identificador, dado que existe mais por termos de logística e não deriva diretamente do processo de adjudicação.

Uma vez que todos os predicados de evolução relativos a este tipo de conhecimento seguem a mesma lógica, o processo apenas será explicado detalhadamente para o caso dos adjudicantes.

Adjudicantes

```
% Inserir adjudicante com nome incerto na base de conhecimento
evolucao(adjudicante(IdAd,nomeIncerto,NIF,Morada)) :-
    evolucaoIncertoInterdito(adjudicante(IdAd,nomeIncerto,NIF,Morada)),
    (demo(adjudicante(IdAd,teste,NIF,Morada),desconhecido); % a excecao ja esta na base
    insercao((excecao(adjudicante(P1,P2,P3,P4)) :- adjudicante(P1,nomeIncerto,P3,P4)))). % se nao estiver, inserir

% Inserir adjudicante com NIF incerto na base de conhecimento
evolucao(adjudicante(IdAd,Nome,nifIncerto,Morada)) :-
    evolucaoIncertoInterdito(adjudicante(IdAd,Nome,nifIncerto,Morada)),
    (demo(adjudicante(IdAd,Nome,teste,Morada),desconhecido);
    insercao((excecao(adjudicante(P1,P2,P3,P4)) :- adjudicante(P1,P2,nifIncerto,P4)))).

% Inserir adjudicante com morada incerta na base de conhecimento
evolucao(adjudicante(IdAd,Nome,NIF,moradaIncerta)) :-
    evolucaoIncertoInterdito(adjudicante(IdAd,Nome,NIF,moradaIncerta)),
    (demo(adjudicante(IdAd,Nome,NIF,teste),desconhecido);
    insercao((excecao(adjudicante(P1,P2,P3,P4)) :- adjudicante(P1,P2,P3,moradaIncerta)))).
```

Figura 36: Predicados de evolução de conhecimento incerto relativo a adjudicantes

Como é possível observar na imagem, estes predicados invocam a evolução do termo com o parâmetro desconhecido através do predicado **evolucaoIncertoInterdito**, demonstrado acima. Se a inserção correr bem e o termo respeitar todos os invariantes, só falta então inserir a respetiva exceção.

Para evitar a repetição de informação, o predicado verifica se já existe uma exceção igual na base de conhecimento, invocando o predicado **demo**, que determina se uma informação é verdadeira, falsa ou desconhecida, e questionando-o com o termo novo, colocando o valor "teste" no campo desconhecido.

```
% Testar conhecimento {V,F,D}
demo(Questao,verdadeiro) :- Questao.
demo(Questao,falso) :- -Questao.
demo(Questao,desconhecido) :- nao(Questao), nao(-Questao).
```

Figura 37: Predicado **demo**

Se a exceção necessária já existir, **demo** determinará o termo como conhecimento desconhecido, caso contrário será conhecimento negativo. Obviamente esta metologia baseia-se na premissa de que o verdadeiro valor dos parâmetros nunca será "teste", pois não faz sentido no contexto do projeto. Conforme o resultado desta inferência, o sistema pode proceder à inserção da exceção no sistema, se não existir, ou não.

Adjudicatárias

```
% Inserir adjudicataria com nome incerto na base de conhecimento
evolucao(adjudicataria(IdAda,nomeIncerto,NIF,Morada)) :-
    evolucaoIncertoInterdito(adjudicataria(IdAda,nomeIncerto,NIF,Morada)),
    (demo(adjudicataria(IdAda,teste,NIF,Morada),desconhecido);
    insercao((excecao(adjudicataria(P1,P2,P3,P4)) :- adjudicataria(P1,nomeIncerto,P3,P4)))).

% Inserir adjudicataria com NIF incerto na base de conhecimento
evolucao(adjudicataria(IdAda,Nome,nifIncerto,Morada)) :-
    evolucaoIncertoInterdito(adjudicataria(IdAda,Nome,nifIncerto,Morada)),
    (demo(adjudicataria(IdAda,Nome,teste,Morada),desconhecido);
    insercao((excecao(adjudicataria(P1,P2,P3,P4)) :- adjudicataria(P1,P2,nifIncerto,P4)))).

% Inserir adjudicataria com morada incerta na base de conhecimento
evolucao(adjudicataria(IdAda,Nome,NIF,moradaIncerta)) :-
    evolucaoIncertoInterdito(adjudicataria(IdAda,Nome,NIF,moradaIncerta)),
    (demo(adjudicataria(IdAda,Nome,NIF,teste),desconhecido);
    insercao((excecao(adjudicataria(P1,P2,P3,P4)) :- adjudicataria(P1,P2,P3,moradaIncerta)))).
```

Figura 38: Predicados de evolução de conhecimento incerto relativo a adjudicatárias

Contratos

Existe um predicado de evolução por cada parâmetro passível de ser desconhecido do contrato, contudo como são todos análogos, vão ser exibidos apenas alguns como exemplo.

```
% Inserir contrato com id do adjudicante incerto na base de conhecimento
evolucao(contrato(IdC,idAdIncerto,IdAda,TC,TP,Desc,V,P,L,D)) :-
    evolucaoIncertoInterdito(contrato(IdC,idAdIncerto,IdAda,TC,TP,Desc,V,P,L,D)),
    (demo(contrato(IdC,teste,IdAda,TC,TP,Desc,V,P,L,D),desconhecido);
    insercao((excecao(contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10)) :- contrato(P1,idAdIncerto,P3,P4,P5,P6,P7,P8,P9,P10)))).

% Inserir contrato com id da adjudicataria incerto na base de conhecimento
evolucao(contrato(IdC,IdAd,idAdaIncerto,TC,TP,Desc,V,P,L,D)) :-
    evolucaoIncertoInterdito(contrato(IdC,IdAd,idAdaIncerto,TC,TP,Desc,V,P,L,D)),
    (demo(contrato(IdC,IdAd,teste,TC,TP,Desc,V,P,L,D),desconhecido);
    insercao((excecao(contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10)) :- contrato(P1,P2,idAdaIncerto,P4,P5,P6,P7,P8,P9,P10)))).

% Inserir contrato de tipo incerto na base de conhecimento
evolucao(contrato(IdC,IdAd,IdAda,tipoIncerto,TP,Desc,V,P,L,D)) :-
    evolucaoIncertoInterdito(contrato(IdC,IdAd,IdAda,tipoIncerto,TP,Desc,V,P,L,D)),
    (demo(contrato(IdC,IdAd,IdAda,teste,TP,Desc,V,P,L,D),desconhecido);
    insercao((excecao(contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10)) :- contrato(P1,P2,P3,tipoIncerto,P5,P6,P7,P8,P9,P10)))).
```

Figura 39: Predicados de evolução de conhecimento incerto relativo a contratos

3.3.3 Conhecimento Imperfeito Interdito

Os predicados de evolução de conhecimento interdito apresentam uma estrutura semelhante à que foi apresentada no capítulo anterior e, tal como no mesmo, seguem todos a mesma linha de pensamento. Por este motivo, será abordado o caso dos adjudicantes em concreto, mais uma vez, e apenas exibidos exemplos dos restantes predicados.

Adjudicantes

```
% Inserir adjudicante com nome interdito na base de conhecimento
evolucao(adjudicante(IdAd,nomeInterdito,NIF,Morada)) :-
    evolucaoIncertoInterdito(adjudicante(IdAd,nomeInterdito,NIF,Morada)),
    (demo(adjudicante(IdAd,teste,NIF,Morada),desconhecido); % a excecao ja esta na base
    insercaoAdjudicanteInterdito((IdAd,nomeInterdito,NIF,Morada),nomeInterdito)).

% Inserir adjudicante com NIF interdito na base de conhecimento
evolucao(adjudicante(IdAd,Nome,nifInterdito,Morada)) :-
    evolucaoIncertoInterdito(adjudicante(IdAd,Nome,nifInterdito,Morada)),
    (demo(adjudicante(IdAd,Nome,teste,Morada),desconhecido);
    insercaoAdjudicanteInterdito((IdAd,Nome,nifInterdito,Morada),nifInterdito)).

% Inserir adjudicante com morada interdita na base de conhecimento
evolucao(adjudicante(IdAd,Nome,NIF,moradaInterdita)) :-
    evolucaoIncertoInterdito(adjudicante(IdAd,Nome,NIF,moradaInterdita)),
    (demo(adjudicante(IdAd,Nome,NIF,teste),desconhecido);
    insercaoAdjudicanteInterdito((IdAd,Nome,NIF,moradaInterdita),moradaInterdita)).
```

Figura 40: Predicados de evolução de conhecimento interdito relativo a adjudicantes

O predicado começa por inserir o termo na base de conhecimento através do **evolucaoIncertoInterdito** (como foi explicado no capítulo 3.2.5, estes dois tipos de conhecimentos obedecem aos mesmos invariantes). De seguida, verifica se a exceção necessária já existe, através do predicado **demo**.

Se a exceção ainda não existir na base, é necessário inserir a mesma, juntamente com a restante informação necessária. Para este efeito, invoca-se o predicado **insercaoAdjudicanteInterdito**.

```
% Inserir as restricoes relacionadas com o novo adjudicante interdito
insercaoAdjudicanteInterdito((IdAd,Nome,NIF,Morada),ParInterdito) :-
    insercao(nuloInterdito(ParInterdito)), % instancia de nulo
    ((ParInterdito = nomeInterdito, insercao((excecao(adjudicante(P1,P2,P3,P4)) :- adjudicante(P1,nomeInterdito,P3,P4))));
     (ParInterdito = nifInterdito, insercao((excecao(adjudicante(P1,P2,P3,P4)) :- adjudicante(P1,P2,nifInterdito,P4))));
     (ParInterdito = moradaInterdita, insercao((excecao(adjudicante(P1,P2,P3,P4)) :- adjudicante(P1,P2,P3,moradaInterdita))))),
    insercao((+adjudicante(P1,P2,P3,P4) :: (solucoes((P1,P2,P3,P4), % invariante
            (adjudicante(IdAd,Nome,NIF,Morada), nao(nuloInterdito(ParInterdito))), R),
            comprimento(R,0)))).
```

Figura 41: Predicado **insercaoAdjudicanteInterdito**

Para começar, insere-se uma instanciação do parâmetro interdito como valor nulo na base. De seguida, o predicado verifica qual dos parâmetros é interdito e introduz a exceção respetiva. Por fim, insere um invariante que impede a evolução de qualquer conhecimento que tente especificar o parâmetro interdito, dando por concluída a evolução do termo.

Adjudicatárias

```
% Inserir as restricoes relacionadas com a nova adjudicataria interdita
insercaoAdjudicatariaInterdita((IdAd,Nome,NIF,Morada),ParInterdito) :-
    insercao(nuloInterdito(ParInterdito)), % instancia de nulo
    ((ParInterdito = nomeInterdito, insercao((excecao(adjudicataria(P1,P2,P3,P4)) :- adjudicataria(P1,nomeInterdito,P3,P4))));
     (ParInterdito = nifInterdito, insercao((excecao(adjudicataria(P1,P2,P3,P4)) :- adjudicataria(P1,P2,nifInterdito,P4))));
     (ParInterdito = moradaInterdita, insercao((excecao(adjudicataria(P1,P2,P3,P4)) :- adjudicataria(P1,P2,P3,moradaInterdita))))),
    insercao((+adjudicataria(P1,P2,P3,P4) :: (solucoes((P1,P2,P3,P4), % invariante
            (adjudicataria(IdAd,Nome,NIF,Morada), nao(nuloInterdito(ParInterdito))), R),
            comprimento(R,0)))).
```

Figura 42: Predicado **insercaoAdjudicatariaInterdita**

```
% Inserir adjudicataria com nome interdito na base de conhecimento
evolucao(adjudicataria(IdAda,nomeInterdito,NIF,Morada)) :-
    evolucaoIncertoInterdito(adjudicataria(IdAda,nomeInterdito,NIF,Morada)),
    (demo(adjudicataria(IdAda,teste,NIF,Morada),desconhecido);
    insercaoAdjudicatariaInterdita((IdAd,nomeInterdito,NIF,Morada),nomeInterdito)).

% Inserir adjudicataria com NIF interdito na base de conhecimento
evolucao(adjudicataria(IdAda,Nome,nifInterdito,Morada)) :-
    evolucaoIncertoInterdito(adjudicataria(IdAda,Nome,nifInterdito,Morada)),
    (demo(adjudicataria(IdAda,Nome,teste,Morada),desconhecido);
    insercaoAdjudicatariaInterdita((IdAd,Nome,nifInterdito,Morada),nifInterdito)).

% Inserir adjudicataria com morada interdita na base de conhecimento
evolucao(adjudicataria(IdAda,Nome,NIF,moradaInterdita)) :-
    evolucaoIncertoInterdito(adjudicataria(IdAda,Nome,NIF,moradaInterdita)),
    (demo(adjudicataria(IdAda,Nome,NIF,teste),desconhecido);
    insercaoAdjudicatariaInterdita((IdAd,Nome,NIF,moradaInterdita),moradaInterdita)).
```

Figura 43: Predicados de evolução de conhecimento interdito relativo a adjudicatárias

Contratos

```
% Inserir as restricoes relacionadas com o novo contrato interdito
insercaoContratoInterdito((IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,D),ParInterdito) :-
    insercao(nuloInterdito(ParInterdito)), % instancia de nulo
    ((ParInterdito = idAdInterdito, insercao((excecao(contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10)) :- contrato(P1,idAdInterdito,P3,P4,P5,P6,P7,P8,P9,P10))));
     (ParInterdito = idAdaInterdito, insercao((excecao(contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10)) :- contrato(P1,P2,idAdaInterdito,P4,P5,P6,P7,P8,P9,P10))));
     (ParInterdito = tipoInterdito, insercao((excecao(contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10)) :- contrato(P1,P2,P3,tipoInterdito,P5,P6,P7,P8,P9,P10))));
     (ParInterdito = procedimentoInterdito, insercao((excecao(contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10)) :- contrato(P1,P2,P3,P4,procedimentoInterdito,P6,P7,P8,P9,P10))));
     (ParInterdito = descricaoInterdita, insercao((excecao(contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10)) :- contrato(P1,P2,P3,P4,P5,descricaoInterdita,P7,P8,P9,P10))));
     (ParInterdito = valorInterdito, insercao((excecao(contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10)) :- contrato(P1,P2,P3,P4,P5,P6,valorInterdito,P8,P9,P10))));
     (ParInterdito = prazoInterdito, insercao((excecao(contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10)) :- contrato(P1,P2,P3,P4,P5,P6,P7,prazoInterdito,P9,P10))));
     (ParInterdito = localInterdito, insercao((excecao(contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10)) :- contrato(P1,P2,P3,P4,P5,P6,P7,P8,localInterdito,P10))));
     (ParInterdito = dataInterdita, insercao((excecao(contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10)) :- contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,dataInterdita))))), % excecao
    insercao((+contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10) :: (solucoes((P1,P2,P3,P4,P5,P6,P7,P8,P9,P10), % invariante
            (contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,D), nao(nuloInterdito(ParInterdito))), R),
            comprimento(R,0)))).
```

Figura 44: Predicado **insercaoContratoInterdito**

Existe um predicado de evolução por cada parâmetro passível de ser desconhecido do contrato, contudo como são todos análogos, vão ser exibidos apenas alguns como exemplo.

```
% Inserir contrato com prazo interdito na base de conhecimento
evolucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,prazoInterdito,L,D)) :-
    evolucaoIncertoInterdito(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,prazoInterdito,L,D)),
    (demo(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,teste,L,D),desconhecido);
    insercaoContratoInterdito((IdC,IdAd,IdAda,TC,TP,Desc,V,prazoInterdito,L,D),prazoInterdito)).

% Inserir contrato com local interdito na base de conhecimento
evolucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,localInterdito,D)) :-
    evolucaoIncertoInterdito(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,localInterdito,D)),
    (demo(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,teste,D),desconhecido);
    insercaoContratoInterdito((IdC,IdAdInterdito,IdAda,TC,TP,Desc,V,P,localInterdito,D),localInterdito)).

% Inserir contrato com data interdita na base de conhecimento
evolucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,dataInterdita)) :-
    evolucaoIncertoInterdito(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,dataInterdita)),
    (demo(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,teste),desconhecido);
    insercaoContratoInterdito((IdC,IdAdInterdito,IdAda,TC,TP,Desc,V,P,L,dataInterdita),dataInterdita)).
```

Figura 45: Predicados de evolução de conhecimento interdito relativo a contratos

3.3.4 Conhecimento Imperfeito Impreciso

Novamente, o grupo precisou de ter em conta a sintaxe deste tipo de conhecimento, explicada e exemplificada em 3.1.2, durante o desenvolvimento de predicados de evolução que possibilitassem a sua introdução na base de conhecimento.

Como resultado, foram elaborados dois tipos de predicados de evolução diferentes para o conhecimento impreciso: o primeiro abrange os casos em que são indicados explicitamente todos os valores possíveis para o parâmetro desconhecido; o segundo trata de situações em que o valor pertence a um certo intervalo.

No primeiro caso, foi criado o predicado **evolucaoImprecisoExplicito**, que reúne os invariantes adequados – neste caso, de termos imprecisos explícitos e termos não negativos - e introduz uma nova exceção relativa ao termo na base de conhecimento. De maneira a poder testar os invariantes, insere também o próprio termo. Se obedecer a todos, remove-o de seguida, deixando apenas a exceção na base, que é o único elemento necessário para a representação deste tipo de conhecimento. Caso contrário, remove ambos, e a evolução de conhecimento falha.

```
% Inserir conhecimento imperfeito impreciso na base de conhecimento
evolucaoImprecisoExplicito(Termo) :-
    solucoes(Inv, (+(excecao(Termo),imprecisoExplicito)::Inv; +(Termo,naoNegativo)::Inv), Lista),
    insercao(excecao(Termo)), insercao(Termo),
    teste(Lista),
    remocao(Termo).
```

Figura 46: Predicado **evolucaoImprecisoExplicito**

No caso dos intervalos, não se pode recorrer à mesma estratégia, dado que estão em questão vários termos simultaneamente (tantos quantos o intervalo comporta), por isso não é viável inserir e testar todos.

Adjudicantes

```
% Inserir adjudicante com NIF entre dois valores
evolucao(adjudicante(IdAd,Nome,(Inf,Sup),Morada)) :-
    invariantesImprecisoImplicito(adjudicante(IdAd,Nome,(Inf,Sup),Morada)),
    insercao((excecao(adjudicante(IdAd,Nome,NIFImpreciso,Morada)) :- NIFImpreciso >= Inf, NIFImpreciso =< Sup)),
    insercao(intervalo(adjudicante,IdAd,nif,(Inf,Sup))).
```

Figura 47: Predicado de evolução de conhecimento impreciso implícito relativo a adjudicantes

Os predicados de evolução de conhecimento impreciso implícito seguem o formato apresentado acima: recebem o predicado com um par (limite inferior, limite superior) do intervalo de valores na posição do argumento desconhecido, e os restantes parâmetros especificados normalmente.

Para começar, recorre-se ao predicado **invariantesImprecisoImplicito** para testar as condições de inserção do termo. Este predicado pode receber adjudicantes, adjudicatárias e contratos, distinguindo as ações a tomar por **pattern matching** do argumento. Atente-se no caso dos adjudicantes:

```
% Verificar a validade dos intervalos de valores imprecisos do adjudicante que se pretender inserir
invariantesImprecisoImplicito(adjudicante(IdAd,Nome,(Inf,Sup),Morada)) :-
    nifValido(Inf), nifValido(Sup), Inf < Sup,
    solucoes(Inv, +(excecao(adjudicante(IdAd,Nome,(Inf,Sup),Morada)),imprecisoImplicito)::Inv, Lista),
    teste(Lista).
```

Figura 48: Predicado **invariantesImprecisoImplicito** relativo a adjudicantes

Em primeiro lugar, o predicado verifica se os limites do intervalo são NIFs válidos, e se o limite inferior é, de facto, inferior ao limite superior. Uma vez verificadas estas condições, filtra os invariantes aplicáveis à inserção de conhecimento impreciso implícito e, neste caso, de exceções de adjudicantes.

Os invariantes deste tipo de conhecimento diferem de todos os outros em dois aspetos:

- Têm em conta que vão receber um par de valores na posição do parâmetro desconhecido, em vez de um só valor;
- Têm em conta que o termo não vai ser inserido na base antes de ser testado com os invariantes, e isto é observável no predicado acima – não há inserção nenhuma, procedendo-se ao teste dos invariantes uma vez reunidos.

Se o novo termo imperfeito passar todas as verificações, o predicado de evolução apresentado na imagem 47 resume a sua execução, introduzindo na base de conhecimento a exceção e uma nova estrutura **intervalo** relativas ao novo termo. Como já foi referido anteriormente, a representação deste tipo de conhecimento só obriga à existência da exceção, sendo que o predicado **intervalo** foi criado pelo grupo para facilitar a gestão posterior e remoção.

Adjudicatárias

Mais uma vez, o processo das adjudicatárias é equivalente ao dos adjudicantes, dado que possuem os mesmos argumentos.

```
% Verificar a validade dos intervalos de valores imprecisos da adjudicataria que se pretender inserir
invariantesImprecisoImplicito(adjudicataria(IdAd,Nome,(Inf,Sup),Morada)) :-
    nifValido(Inf), nifValido(Sup), Inf < Sup,
    solucoes(Inv, +(excecao(adjudicataria(IdAd,Nome,(Inf,Sup),Morada)),imprecisoImplicito)::Inv, Lista),
    teste(Lista).
```

Figura 49: Predicado **invariantesImprecisoImplicito** relativo a adjudicatárias

```
% Inserir adjudicataria com NIF entre dois valores
evolucao(adjudicataria(IdAda,Nome,(Inf,Sup),Morada)) :-
    invariantesImprecisoImplicito(adjudicataria(IdAda,Nome,(Inf,Sup),Morada)),
    insercao((excecao(adjudicataria(IdAda,Nome,NIFImpreciso,Morada)) :- NIFImpreciso >= Inf, NIFImpreciso =< Sup)),
    insercao(intervalo(adjudicataria,IdAda,nif,(Inf,Sup))).
```

Figura 50: Predicado de evolução de conhecimento impreciso implícito relativo a adjudicatárias

Contratos

Para os contratos, foi necessário desenvolver um predicado de evolução, e respetivo cenário do **invariantesImprecisoImplicito**, para cada parâmetro possivelmente impreciso – valor, prazo e ano, mês e dia da data. Todos estes parâmetros constituem valores numéricos e podem pertencer a um intervalo.

Analisar-se-á o exemplo do valor do contrato, sendo que todos os outros seguem a mesma lógica.

```
% Inserir contrato com valor entre dois valores
evolucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,(Inf,Sup),P,L,D)) :-
    invariantesImprecisoImplicito(contrato(IdC,IdAd,IdAda,TC,TP,Desc,(Inf,Sup),P,L,D)),
    insercaoContratoImprecisoImplicito((IdC,IdAd,IdAda,TC,TP,Desc,VImpreciso,P,L,D),VImpreciso,valor,Inf,Sup).
```

Figura 51: Predicado de evolução de contratos com valor pertencente a um intervalo

Começa-se por invocar a função **invariantesImprecisoImplicito** relativa a valores de contrato imprecisos:

```
% Verificar a validade dos intervalos de valores imprecisos do contrato que se pretender inserir
invariantesImprecisoImplicito(contrato(IdC,IdAd,IdAda,TC,TP,Desc,(Inf,Sup),P,L,D)) :-
    numeroValido(Inf), numeroValido(Sup), Inf < Sup,
    solucoes(Inv, +(contrato(IdC,IdAd,IdAda,TC,TP,Desc,Inf,P,L,D),naoNegativo)::Inv, ListaNaoNegativoInf),
    teste(ListaNaoNegativoInf),
    solucoes(Inv, +(contrato(IdC,IdAd,IdAda,TC,TP,Desc,Sup,P,L,D),naoNegativo)::Inv, ListaNaoNegativoSup),
    teste(ListaNaoNegativoSup),
    solucoes(Inv, +(excecao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,(Inf,Sup),P,L,D)),imprecisoImplicito)::Inv, ListaImprecisoImplicito),
    teste(ListaImprecisoImplicito).
```

Figura 52: Predicado **invariantesImprecisoImplicito** relativo a contratos

Em primeiro lugar, é necessário verificar se os limites dados são **números** válidos e se o inferior é menor que o superior. De seguida, reúne-se os invariantes relativos à inserção de conhecimento **não negativo** e, neste caso, aplicáveis a contratos. Procede-se ao teste de um contrato (não presente na base de conhecimento) cujo valor é o limite inferior do intervalo, e repete-se o processo para um contrato com o limite superior.

Para além do intervalo de valores em si, é necessário garantir que todos os outros parâmetros do contrato também são válidos – tarefa desempenhada pelos invariantes de conhecimento não negativo. No que toca a adjudicantes e adjudicatárias, apenas existem tais restrições sobre o NIF, pelo que se testa diretamente no predicado **invariantesImprecisosImplicitos**, em vez de filtrar os invariantes **nãoNegativo**. Contudo, os contratos possuem bastantes mais parâmetros, pelo que essa estratégia deixa de ser viável e se recorre ao processo descrito acima.

Se os parâmetros dos contratos forem válidos, resta ainda verificar as condições existentes no que toca ao próprio conhecimento impreciso implícito. Como tal, o predicado filtra esses mesmos invariantes e testa-os.

Uma vez validados todos os requerimentos, o predicado de evolução invoca o predicado **insercaoContratoImprecisoImplicito** para inserir a exceção relativa ao novo termo e a respetiva estrutura **intervalo**.

```
insercaoContratoImprecisoImplicito((IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,D),ParImpreciso,NomePar,Inf,Sup) :-
    insercao((excecao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,D)) :- ParImpreciso >= Inf, ParImpreciso =< Sup)),
    insercao(intervalo(contrato,IdC,NomePar,(Inf,Sup))).
```

Figura 53: Predicado **insercaoContratoImprecisoImplicito**

3.4 Involução de Conhecimento

Tal como foram elaborados predicados para tratar da verificação e inserção de informação na base de conhecimento, foram também criados predicados de involução, que fazem o oposto.

Estes predicados são, de certa forma, mais simples, uma vez que existem poucos invariantes de remoção, e só no caso de conhecimento perfeito positivo.

3.4.1 Conhecimento Perfeito

O predicado de involução de conhecimento perfeito positivo segue uma lógica análoga ao respetivo predicado de evolução, embora oposta - **remoção** e não inserção.

```
% Remover conhecimento perfeito positivo da base de conhecimento
involucaoPos(Termo) :-
    solucoes(Invariante, -(Termo,positivo)::Invariante, Lista),
    remocao(Termo),
    teste(Lista).
```

Figura 54: Predicado de involução de conhecimento positivo

Começa por filtrar os invariantes de remoção de termos positivos, procedendo à remoção do item e, posteriormente, ao teste dos invariantes. Caso não seja possível remover o termo, o mesmo é inserido de volta na base de conhecimento, e a involução falha.

No que toca a conhecimento perfeito negativo, o processo é exatamente o mesmo, sendo a única diferença o facto de os invariantes filtrados dizerem respeito a conhecimento negativo.

```
% Remover conhecimento perfeito negativo da base de conhecimento
involucaoNeg(Termo) :-
    solucoes(Invariante, -(Termo,negativo)::Invariante, Lista),
    remocao(Termo),
    teste(Lista).
```

Figura 55: Predicado de involução de conhecimento negativo

Porém, este tipo de conhecimento pefeito possui ainda mais uma nuance que o positivo. Enquanto que os termos positivos da base de conhecimento têm garantidamente identificadores únicos, isto não é o caso no conhecimento negativo. Tendo em conta que termos negativos constituem factos falsos, é possível ter um termo correspondente a uma adjudicatária de identificador 3 a indicar que não se chama Município de Amares e outro a constatar que também não se chama Município de Ovar.

Como tal, e dado que foi implementada opção de remoção de termos por identificador, como será explicado mais à frente neste relatório, surgiu a necessidade de elaborar um predi-

cado que, dada uma lista com todos os termos negativos em questão, tratasse da sua involução individual e sequencialmente.

```
% Pode haver varios predicados negativos com o mesmo id, logo remover uma
% entidade com um certo id implica a remocao de todos
involucaoListaNeg([]).
involucaoListaNeg([H|T]) :- involucaoNeg(H), involucaoListaNeg(T).
```

Figura 56: Predicado de involução de uma lista de termos negativos

3.4.2 Conhecimento Imperfeito Incerto

No caso do conhecimento imperfeito, a única condição para a remoção bem-sucedida do termo é a existência do mesmo na base. O processo é análogo para adjudicantes, adjudicatárias e contratos, por isso será analisado um predicado específico e mostrados exemplos dos outros.

Tal como na evolução, foi necessário elaborar um predicado por cada parâmetro passível de ser incerto, para os três predicados estruturantes do sistema. O sistema de inferência invoca o predicado de involução adequado à situação em questão através de **pattern matching**.

Adjudicantes

```
% Remover adjudicante com nome incerto na base de conhecimento
involucao(adjudicante(IdAd,nomeIncerto,NIF,Morada)) :-
    remocao(adjudicante(IdAd,nomeIncerto,NIF,Morada)),
    ((solucoes(adjudicante(_,nomeIncerto,_,_), adjudicante(_,nomeIncerto,_,_), R), comprimento(R,N), N \= 0);
    remocao((excecao(adjudicante(P1,P2,P3,P4)) :- adjudicante(P1,nomeIncerto,P3,P4)))).

% Remover adjudicante com NIF incerto na base de conhecimento
involucao(adjudicante(IdAd,Nome,nifIncerto,Morada)) :-
    remocao(adjudicante(IdAd,Nome,nifIncerto,Morada)),
    ((solucoes(adjudicante(_,_,nifIncerto,_), adjudicante(_,_,nifIncerto,_), R), comprimento(R,N), N \= 0);
    remocao((excecao(adjudicante(P1,P2,P3,P4)) :- adjudicante(P1,P2,nifIncerto,P4)))).

% Remover adjudicante com morada incerta na base de conhecimento
involucao(adjudicante(IdAd,Nome,NIF,moradaIncerta)) :-
    remocao(adjudicante(IdAd,Nome,NIF,moradaIncerta)),
    ((solucoes(adjudicante(_,_,_,moradaIncerta), adjudicante(_,_,_,moradaIncerta), R), comprimento(R,N), N \= 0);
    remocao((excecao(adjudicante(P1,P2,P3,P4)) :- adjudicante(P1,P2,P3,moradaIncerta)))).
```

Figura 57: Predicados de involução de conhecimento incerto relativo a adjudicantes

Como é possível observar na imagem, o predicado começa por remover o adjudicante. De seguida, de maneira a fazer uma boa gestão da informação na base de conhecimento, verifica se ainda existe mais algum adjudicante com nome incerto. Como foi abordado no capítulo 3.3.2, não existem exceções repetidas, dado que se verifica se já existe alguma igual ou não no momento da inserção.

Como tal, se o termo removido for o único adjudicante de nome incerto, o predicado remove também a exceção correspondente, uma vez que não ficaria a servir nenhum propósito na base de conhecimento. Caso ainda existam mais termos desse tipo, remover a exceção implicaria que os mesmos deixariam de ser considerados desconhecidos pelo sistema de inferência e passariam a ser falsos (adjudicantes com os mesmo parâmetros e algum nome especificado), pelo que não se extrai a exceção.

Adjudicatárias

```
% Remover adjudicataria com nome incerto na base de conhecimento
involucao(adjudicataria(IdAda,nomeIncerto,NIF,Morada)) :-
    remocao(adjudicataria(IdAda,nomeIncerto,NIF,Morada)),
    ((solucoes(adjudicataria(_,nomeIncerto,_,_), adjudicataria(_,nomeIncerto,_,_), R), comprimento(R,N), N \= 0);
    remocao((excecao(adjudicataria(P1,P2,P3,P4)) :- adjudicataria(P1,nomeIncerto,P3,P4)))).

% Remover adjudicataria com NIF incerto na base de conhecimento
involucao(adjudicataria(IdAda,Nome,nifIncerto,Morada)) :-
    remocao(adjudicataria(IdAda,Nome,nifIncerto,Morada)),
    ((solucoes(adjudicataria(_,_,nifIncerto,_), adjudicataria(_,_,nifIncerto,_), R), comprimento(R,N), N \= 0);
    remocao((excecao(adjudicataria(P1,P2,P3,P4)) :- adjudicataria(P1,P2,nifIncerto,P4)))).

% Remover adjudicataria com morada incerta na base de conhecimento
involucao(adjudicataria(IdAda,Nome,NIF,moradaIncerta)) :-
    remocao(adjudicataria(IdAda,Nome,NIF,moradaIncerta)),
    ((solucoes(adjudicataria(_,_,_,moradaIncerta), adjudicataria(_,_,_,moradaIncerta), R), comprimento(R,N), N \= 0);
    remocao((excecao(adjudicataria(P1,P2,P3,P4)) :- adjudicataria(P1,P2,P3,moradaIncerta)))).
```

Figura 58: Predicados de involução de conhecimento incerto relativo a adjudicatárias

Contratos

Serão mostrados apenas alguns exemplos dos predicados de evolução de contratos, tendo em conta que os restantes são análogos.

```
% Remover contrato com id do adjudicante incerto na base de conhecimento
involucao(contrato(IdC,idAdIncerto,IdAda,TC,TP,Desc,V,P,L,D)) :-
    remocao(contrato(IdC,idAdIncerto,IdAda,TC,TP,Desc,V,P,L,D)),
    ((solucoes(contrato(_,idAdIncerto,_,_,_,_,_,_,_,_), contrato(_,idAdIncerto,_,_,_,_,_,_,_,_), R), comprimento(R,N), N \= 0);
    remocao((excecao(contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10)) :- contrato(P1,idAdIncerto,P3,P4,P5,P6,P7,P8,P9,P10)))).

% Remover contrato com id da adjudicataria incerto na base de conhecimento
involucao(contrato(IdC,IdAd,idAdaIncerto,TC,TP,Desc,V,P,L,D)) :-
    remocao(contrato(IdC,IdAd,idAdaIncerto,TC,TP,Desc,V,P,L,D)),
    ((solucoes(contrato(_,_,idAdaIncerto,_,_,_,_,_,_,_), contrato(_,_,idAdaIncerto,_,_,_,_,_,_,_), R), comprimento(R,N), N \= 0);
    remocao((excecao(contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10)) :- contrato(P1,P2,idAdaIncerto,P4,P5,P6,P7,P8,P9,P10)))).

% Remover contrato de tipo incerto na base de conhecimento
involucao(contrato(IdC,IdAd,IdAda,tipoIncerto,TP,Desc,V,P,L,D)) :-
    remocao(contrato(IdC,IdAd,IdAda,tipoIncerto,TP,Desc,V,P,L,D)),
    ((solucoes(contrato(_,_,_,tipoIncerto,_,_,_,_,_,_), contrato(_,_,_,tipoIncerto,_,_,_,_,_,_), R), comprimento(R,N), N \= 0);
    remocao((excecao(contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10)) :- contrato(P1,P2,P3,tipoIncerto,P5,P6,P7,P8,P9,P10)))).
```

Figura 59: Predicados de involução de conhecimento incerto relativo a contratos

3.4.3 Conhecimento Imperfeito Interdito

Adjudicantes

Os predicados de involução de adjudicantes com parâmetros interditos começam por invocar o predicado **remocaoAdjudicanteInterdito**, que é responsável por remover o adjudicante e respetivo invariante da base de conhecimento.

```
% Remover o termo em questao e respetivo invariante do adjudicante
remocaoAdjudicanteInterdito((IdAd,Nome,NIF,Morada),ParInterdito) :-
    remocao(adjudicante(IdAd,Nome,NIF,Morada)), % termo
    remocao((+adjudicante(P1,P2,P3,P4) :: (solucoes((P1,P2,P3,P4), % invariante
            (adjudicante(IdAd,Nome,NIF,Morada), nao(nuloInterdito(ParInterdito))), R),
            comprimento(R,0)))).
```

Figura 60: Predicado **remocaoAdjudicanteInterdito**

De seguida, verificam se o termo removido era o único na base de conhecimento com o parâmetro interdito em questão. Se for o caso, procedem à remoção da exceção correspondente, bem como a instância do parâmetro interdito como termo nulo. Caso contrário, o predicado de involução dá-se por concluído com sucesso sem executar este último passo, uma vez que isso invalidaria os restantes termos descritos existentes.

```
% Remover adjudicante com nome interdito na base de conhecimento
involucao(adjudicante(IdAd,nomeInterdito,NIF,Morada)) :-
    remocaoAdjudicanteInterdito((IdAd,nomeInterdito,NIF,Morada),nomeInterdito),
    ((solucoes(adjudicante(_,nomeInterdito,_,_), adjudicante(_,nomeInterdito,_,_), R), comprimento(R,N), N \= 0);
    remocao((excecao(adjudicante(P1,P2,P3,P4)) :- adjudicante(P1,nomeInterdito,P3,P4))),
    remocao(nuloInterdito(nomeInterdito))).

% Remover adjudicante com NIF interdito na base de conhecimento
involucao(adjudicante(IdAd,Nome,nifInterdito,Morada)) :-
    remocaoAdjudicanteInterdito((IdAd,Nome,nifInterdito,Morada),nifInterdito),
    ((solucoes(adjudicante(_,_,nifInterdito,_), adjudicante(_,_,nifInterdito,_), R), comprimento(R,N), N \= 0);
    remocao((excecao(adjudicante(P1,P2,P3,P4)) :- adjudicante(P1,P2,nifIncerto,P4))),
    remocao(nuloInterdito(nifInterdito))).

% Remover adjudicante com morada interdita na base de conhecimento
involucao(adjudicante(IdAd,Nome,NIF,moradaInterdita)) :-
    remocaoAdjudicanteInterdito((IdAd,Nome,NIF,moradaInterdita),moradaInterdita),
    ((solucoes(adjudicante(_,_,_,moradaInterdita), adjudicante(_,_,_,moradaInterdita), R), comprimento(R,N), N \= 0);
    remocao((excecao(adjudicante(P1,P2,P3,P4)) :- adjudicante(P1,P2,P3,moradaInterdita))),
    remocao(nuloInterdito(moradaInterdita))).
```

Figura 61: Predicados de involução de conhecimento interdito relativos a adjudicantes

O processo das adjudicatárias e dos contratos é análogo, como é possível observar nos exemplos fornecidos abaixo.

Adjudicatárias

```
% Remover o termo em questao e respetivo invariante da adjudicataria
remocaoAdjudicatariaInterdita((IdAd,Nome,NIF,Morada),ParInterdito) :-
    remocao(adjudicataria(IdAd,Nome,NIF,Morada)), % termo
    remocao((+adjudicataria(P1,P2,P3,P4) :: (solucoes((P1,P2,P3,P4), % invariante
            (adjudicataria(IdAd,Nome,NIF,Morada), nao(nuloInterdito(ParInterdito))), R),
            comprimento(R,0)))).
```

Figura 62: Predicado **remocaoAdjudicatariaInterdita**

```
% Remover adjudicataria com nome interdito na base de conhecimento
involucao(adjudicataria(IdAd,nomeInterdito,NIF,Morada)) :-
    remocaoAdjudicatariaInterdita((IdAd,nomeInterdito,NIF,Morada),nomeInterdito),
    ((solucoes(adjudicataria(_,nomeInterdito,_,_), adjudicataria(_,nomeInterdito,_,_), R), comprimento(R,N), N \= 0);
    remocao((excecao(adjudicataria(P1,P2,P3,P4)) :- adjudicataria(P1,nomeInterdito,P3,P4))),
    remocao(nuloInterdito(nomeInterdito))).

% Remover adjudicataria com NIF interdito na base de conhecimento
involucao(adjudicataria(IdAd,Nome,nifInterdito,Morada)) :-
    remocaoAdjudicatariaInterdita((IdAd,Nome,nifInterdito,Morada),nifInterdito),
    ((solucoes(adjudicataria(_,_,nifInterdito,_), adjudicataria(_,_,nifInterdito,_), R), comprimento(R,N), N \= 0);
    remocao((excecao(adjudicataria(P1,P2,P3,P4)) :- adjudicataria(P1,P2,nifIncerto,P4))),
    remocao(nuloInterdito(nifInterdito))).

% Remover adjudicataria com morada interdita na base de conhecimento
involucao(adjudicataria(IdAd,Nome,NIF,moradaInterdita)) :-
    remocaoAdjudicatariaInterdita((IdAd,Nome,NIF,moradaInterdita),moradaInterdita),
    ((solucoes(adjudicataria(_,_,_,moradaInterdita), adjudicataria(_,_,_,moradaInterdita), R), comprimento(R,N), N \= 0);
    remocao((excecao(adjudicataria(P1,P2,P3,P4)) :- adjudicataria(P1,P2,P3,moradaInterdita))),
    remocao(nuloInterdito(moradaInterdita))).
```

Figura 63: Predicados de involução de conhecimento interdito relativos a adjudicatárias

Contratos

```
% Remover o termo em questao e respetivo invariante do contrato
remocaoContratoInterdito((IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,D),ParInterdito) :-
    remocao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,D)),
    remocao((+contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10) :: (solucoes((P1,P2,P3,P4,P5,P6,P7,P8,P9,P10),
            (contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,D), nao(nuloInterdito(ParInterdito))), R),
            comprimento(R,0)))).
```

Figura 64: Predicado **remocaoContratoInterdito**

Tal como foi feito em 3.4.2, os predicados de involução de contratos deste tipo de conhecimento não serão exibidos na sua totalidade, dado que seguem todos a mesma linha de raciocínio.

```
% Remover contrato com prazo interdito na base de conhecimento
involucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,prazoInterdito,L,D)) :-
    remocaoContratoInterdito((IdC,IdAd,IdAda,TC,TP,Desc,V,prazoInterdito,L,D),prazoInterdito),
    ((solucoes(contrato(_,_,_,_,_,_,_,_,prazoInterdito,_,_), contrato(_,_,_,_,_,_,_,_,prazoInterdito,_,_), R), comprimento(R,N), N \= 0);
    remocao((excecao(contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10)) :- contrato(P1,P2,P3,P4,P5,P6,P7,prazoInterdito,P9,P10))),
    remocao(nuloInterdito(prazoInterdito))).

% Remover contrato com local interdito na base de conhecimento
involucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,localInterdito,D)) :-
    remocaoContratoInterdito((IdC,IdAd,IdAda,TC,TP,Desc,V,P,localInterdito,D),localInterdito),
    ((solucoes(contrato(_,_,_,_,_,_,_,_,localInterdito,_), contrato(_,_,_,_,_,_,_,_,localInterdito,_), R), comprimento(R,N), N \= 0);
    remocao((excecao(contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10)) :- contrato(P1,P2,P3,P4,P5,P6,P7,P8,localInterdito,P10))),
    remocao(nuloInterdito(localInterdito))).

% Remover contrato com data interdita na base de conhecimento
involucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,dataInterdita)) :-
    remocaoContratoInterdito((IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,dataInterdita),dataInterdita),
    ((solucoes(contrato(_,_,_,_,_,_,_,_,_,dataInterdita), contrato(_,_,_,_,_,_,_,_,_,dataInterdita), R), comprimento(R,N), N \= 0);
    remocao((excecao(contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10)) :- contrato(P1,P2,P3,P4,P5,P6,P7,P8,P9,dataInterdita))),
    remocao(nuloInterdito(dataInterdita))).
```

Figura 65: Predicados de involução de conhecimento interdito relativos a contratos

3.4.4 Conhecimento Imperfeito Impreciso

Foi necessário desenvolver dois métodos de involução diferentes para remover conhecimento impreciso explícito e implícito.

No caso do primeiro, como já foi explicado no capítulo 3.2, é possível existirem vários predicados do mesmo tipo (p.e. adjudicantes) com o mesmo identificador, apenas variando um dos restantes parâmetros entre eles. Logo, tal como é o caso do conhecimento perfeito negativo, a involução de um termo deste tipo de conhecimento, com um determinado identificador, implica a remoção de todas as alternativas existentes.

Como tal, foi elaborado o predicado **involucaoImprecisoExplicito**, que recebe a lista dos termos em questão e os remove sequencialmente da base de conhecimento.

```
% Pode haver varios predicados imprecisos com o mesmo id, logo remover uma
% entidade com um certo id implica a remocao de todos
involucaoImprecisoExplicito([]).
involucaoImprecisoExplicito([H|T]) :- remocao(excecao(H)), involucaoImprecisoExplicito(T).
```

Figura 66: Predicados de involução de listas de termos imprecisos explícitos

Já no caso de conhecimento imperfeito implícito, a operação é mais simples. Os predicados desenvolvidos recebem, como argumento, a estrutura **intervalo** correspondente ao termo que se pretende remover – especificando o tipo de predicado, o seu identificador, o nome do parâmetro desconhecido e os limites do intervalo.

Com estes dados, executa a remoção da exceção relativa ao intervalo, e remove também o predicado intervalo em questão, como é possível observar nos exemplos abaixo, que são apenas alguns dos predicados elaborados:

```
% Remover adjudicante com NIF entre dois valores
involucao(intervalo(Predicado,IdAd,Parametro,(Inf,Sup))) :-
    remocao((excecao(adjudicante(IdAd,_,NIF,_)) :- NIF >= Inf, NIF =< Sup)),
    remocao(intervalo(Predicado,IdAd,Parametro,(Inf,Sup))).

% Remover adjudicataria com NIF entre dois valores
involucao(intervalo(Predicado,IdAda,Parametro,(Inf,Sup))) :-
    remocao((excecao(adjudicataria(IdAda,_,NIF,_)) :- NIF >= Inf, NIF =< Sup)),
    remocao(intervalo(Predicado,IdAda,Parametro,(Inf,Sup))).

% Remover contrato com valor entre dois valores
involucao(intervalo(Predicado,IdC,valor,(Inf,Sup))) :-
    remocao((excecao(contrato(IdC,_,_,_,_,_,V,_,_,_)) :- V >= Inf, V =< Sup)),
    remocao(intervalo(Predicado,IdC,valor,(Inf,Sup))).

% Remover contrato com prazo entre dois valores
involucao(intervalo(Predicado,IdC,prazo,(Inf,Sup))) :-
    remocao((excecao(contrato(IdC,_,_,_,_,_,_,P,_,_)) :- P >= Inf, P =< Sup)),
    remocao(intervalo(Predicado,IdC,prazo,(Inf,Sup))).
```

Figura 67: Predicados de involução de conhecimento impreciso implícito

3.5 Operações do utilizador

Uma vez compilado/consultado o código, o utilizador tem liberdade para executar qualquer predicado do programa. Contudo, partindo do princípio que nem toda a gente que poderá recorrer a este sistema de inferência será necessariamente um perito em *PROLOG*, e entenderá as nuances da linguagem, ou mesmo alguma da sua sintaxe mais complexa, o grupo encarregou-se de elaborar um conjunto de predicados para facilitar o papel do utilizador.

3.5.1 Gestão da Base de Conhecimento

Como foi abordado nas secções 3.3 e 3.4, a evolução e involução de conhecimento é capazes de atingir um grande nível de complexidade. Por este motivo, o grupo desenvolveu um conjunto de predicados para facilitar estas operações.

Estão definidos no ficheiro principal do projeto, **tp1.pl**, a partir do qual deve ser compilado o sistema, uma vez que se encarrega do estabelecimento da ligação aos restantes ficheiros.

De maneira a generalizar as operações de gestão de conhecimento e os vários tipos de predicados de evolução/involução existentes, o grupo elaborou apenas três predicados para a inserção de conhecimento - **inserirAdjudicante**, **inserirAdjudicataria** e **inserirContrato** - e outros três para a remoção - **removerAdjudicante**, **removerAdjudicataria** e **removerContrato**.

Desta maneira, o utilizador apenas necessita de prestar atenção ao formato em que deve inserir os argumentos, para o tipo de conhecimento que pretender, que o sistema de inferência trata do resto, invocando o respetivo predicado de evolução/involução.

Inserção de Conhecimento Perfeito

Para inserir conhecimento perfeito, o primeiro argumento fornecido deve distinguir se se trata de conhecimento positivo ou negativo, e o segundo argumento deve ser o conjunto dos parâmetros do novo termo, entre parênteses.

```
% ----- Perfeito ----
% O primeiro argumento deve indicar se o conhecimento e positivo ou negativo

inserirAdjudicante(pos,(IdAd,N,NIF,M)) :- evolucaoPos(adjudicante(IdAd,N,NIF,M)).
inserirAdjudicante(neg,(IdAd,N,NIF,M)) :- evolucaoNeg(adjudicante(IdAd,N,NIF,M)).

inserirAdjudicataria(pos,(IdAda,N,NIF,M)) :- evolucaoPos(adjudicataria(IdAda,N,NIF,M)).
inserirAdjudicataria(neg,(IdAda,N,NIF,M)) :- evolucaoNeg(adjudicataria(IdAda,N,NIF,M)).

inserirContrato(pos,(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,D)) :- evolucaoPos(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,D)).
inserirContrato(neg,(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,D)) :- evolucaoNeg(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,D)).
```

Figura 68: Predicados de inserção de conhecimento perfeito

Inserção de Conhecimento Imperfeito Incerto

No caso de conhecimento incerto, apenas é necessário fornecer ao predicado os parâmetros da nova entidade. Uma vez que um deles vai ser incerto, o sistema de inferência é capaz de

reconhecer o caso em questão por **pattern matching** e invocar o respetivo predicado de evolução.

Dito isto, é preciso ter cuidado para escrever corretamente a palavra-chave relativo ao parâmetro incerto, caso contrário o sistema não reconhecerá a operação.

```
% ----- Imperfeito Incerto ----
% Deve usar as keywords abaixo correspondentes ao parametro incerto que se pretende inserir, caso contrario nao funcionara

inserirAdjudicante(IdAd,nomeIncerto,NIF,M) :- evolucao(adjudicante(IdAd,nomeIncerto,NIF,M)).
inserirAdjudicante(IdAd,N,nifIncerto,M) :- evolucao(adjudicante(IdAd,N,nifIncerto,M)).
inserirAdjudicante(IdAd,N,NIF,moradaIncerta) :- evolucao(adjudicante(IdAd,N,NIF,moradaIncerta)).

inserirAdjudicataria(IdAda,nomeIncerto,NIF,M) :- evolucao(adjudicataria(IdAda,nomeIncerto,NIF,M)).
inserirAdjudicataria(IdAda,N,nifIncerto,M) :- evolucao(adjudicataria(IdAda,N,nifIncerto,M)).
inserirAdjudicataria(IdAda,N,NIF,moradaIncerta) :- evolucao(adjudicataria(IdAda,N,NIF,moradaIncerta)).

inserirContrato(IdC,idAdIncerto,IdAda,TC,TP,Desc,V,P,L,D) :- evolucao(contrato(IdC,idAdIncerto,IdAda,TC,TP,Desc,V,P,L,D)).
inserirContrato(IdC,IdAd,idAdaIncerto,TC,TP,Desc,V,P,L,D) :- evolucao(contrato(IdC,IdAd,idAdaIncerto,TC,TP,Desc,V,P,L,D)).
inserirContrato(IdC,IdAd,IdAda,tipoIncerto,TP,Desc,V,P,L,D) :- evolucao(contrato(IdC,IdAd,IdAda,tipoIncerto,TP,Desc,V,P,L,D)).
inserirContrato(IdC,IdAd,IdAda,TC,procedimentoIncerto,Desc,V,P,L,D) :- evolucao(contrato(IdC,IdAd,IdAda,TC,procedimentoIncerto,Desc,V,P,L,D)).
inserirContrato(IdC,IdAd,IdAda,TC,TP,descricaoIncerta,V,P,L,D) :- evolucao(contrato(IdC,IdAd,IdAda,TC,TP,descricaoIncerta,V,P,L,D)).
inserirContrato(IdC,IdAd,IdAda,TC,TP,Desc,valorIncerto,P,L,D) :- evolucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,valorIncerto,P,L,D)).
inserirContrato(IdC,IdAd,IdAda,TC,TP,Desc,V,prazoIncerto,L,D) :- evolucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,prazoIncerto,L,D)).
inserirContrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,localIncerto,D) :- evolucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,localIncerto,D)).
inserirContrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,dataIncerta) :- evolucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,dataIncerta)).
```

Figura 69: Predicados de inserção de conhecimento incerto

Inserção de Conhecimento Imperfeito Interdito

A introdução de conhecimento interdito é análoga à de conhecimento incerto.

```
% ----- Imperfeito Interdito ----
% Deve usar as keywords abaixo correspondentes ao parametro interdito que se pretende inserir

inserirAdjudicante(IdAd,nomeInterdito,NIF,M) :- evolucao(adjudicante(IdAd,nomeInterdito,NIF,M)).
inserirAdjudicante(IdAd,N,nifInterdito,M) :- evolucao(adjudicante(IdAd,N,nifInterdito,M)).
inserirAdjudicante(IdAd,N,NIF,moradaInterdita) :- evolucao(adjudicante(IdAd,N,NIF,moradaInterdita)).

inserirAdjudicataria(IdAda,nomeInterdito,NIF,M) :- evolucao(adjudicataria(IdAda,nomeInterdito,NIF,M)).
inserirAdjudicataria(IdAda,N,nifInterdito,M) :- evolucao(adjudicataria(IdAda,N,nifInterdito,M)).
inserirAdjudicataria(IdAda,N,NIF,moradaInterdita) :- evolucao(adjudicataria(IdAda,N,NIF,moradaInterdita)).

inserirContrato(IdC,idAdInterdito,IdAda,TC,TP,Desc,V,P,L,D) :- evolucao(contrato(IdC,idAdInterdito,IdAda,TC,TP,Desc,V,P,L,D)).
inserirContrato(IdC,IdAd,idAdaInterdito,TC,TP,Desc,V,P,L,D) :- evolucao(contrato(IdC,IdAd,idAdaInterdito,TC,TP,Desc,V,P,L,D)).
inserirContrato(IdC,IdAd,IdAda,tipoInterdito,TP,Desc,V,P,L,D) :- evolucao(contrato(IdC,IdAd,IdAda,tipoInterdito,TP,Desc,V,P,L,D)).
inserirContrato(IdC,IdAd,IdAda,TC,procedimentoInterdito,Desc,V,P,L,D) :- evolucao(contrato(IdC,IdAd,IdAda,TC,procedimentoInterdito,Desc,V,P,L,D)).
inserirContrato(IdC,IdAd,IdAda,TC,TP,descricaoInterdita,V,P,L,D) :- evolucao(contrato(IdC,IdAd,IdAda,TC,TP,descricaoInterdita,V,P,L,D)).
inserirContrato(IdC,IdAd,IdAda,TC,TP,Desc,valorInterdito,P,L,D) :- evolucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,valorInterdito,P,L,D)).
inserirContrato(IdC,IdAd,IdAda,TC,TP,Desc,V,prazoInterdito,L,D) :- evolucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,prazoInterdito,L,D)).
inserirContrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,localInterdito,D) :- evolucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,localInterdito,D)).
inserirContrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,dataInterdita) :- evolucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,dataInterdita)).
```

Figura 70: Predicados de inserção de conhecimento interdito

Inserção de Conhecimento Imperfeito Impreciso

Para inserir conhecimento impreciso **explícito**, o formato dos argumentos é semelhante ao de conhecimento perfeito: o primeiro argumento deve ser a palavra "impreciso", indicando o tipo de conhecimento, e o segundo argumento o conjunto de parâmetros do novo termo.

```
% Predicados completos que constituem conhecimento impreciso
inserirAdjudicante(impreciso,(IdAd,N,NIF,M)) :- evolucaoImprecisoExplicito(adjudicante(IdAd,N,NIF,M)).
inserirAdjudicataria(impreciso,(IdAda,N,NIF,M)) :- evolucaoImprecisoExplicito(adjudicataria(IdAda,N,NIF,M)).
inserirContrato(impreciso,(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,D)) :- evolucaoImprecisoExplicito(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,D)).
```

Figura 71: Predicados de inserção de conhecimento impreciso explícito

No caso de conhecimento impreciso **implícito**, o utilizador deve fornecer os parâmetros da entidade da introduzir, exceto que no lugar do parâmetro impreciso deve indicar um par de valores - os limites do intervalo.

```
% Predicados que possuem parametros entre certos valores
% Adjudicante - NIF
inserirAdjudicante(IdAd,N,(Inf,Sup),M) :- evolucao(adjudicante(IdAd,N,(Inf,Sup),M)).
% Adjudicataria - NIF
inserirAdjudicataria(IdAda,N,(Inf,Sup),M) :- evolucao(adjudicataria(IdAda,N,(Inf,Sup),M)).
% Contrato - Valor
inserirContrato(IdC,IdAd,IdAda,TC,TP,Desc,(Inf,Sup),P,L,D) :- evolucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,(Inf,Sup),P,L,D)).
% Contrato - Prazo
inserirContrato(IdC,IdAd,IdAda,TC,TP,Desc,V,(Inf,Sup),L,D) :- evolucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,(Inf,Sup),L,D)).
% Contrato - Ano da data
inserirContrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,(data((Inf,Sup),Mes,Dia))) :- evolucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,data((Inf,Sup),Mes,Dia))).
% Contrato - Mes da data
inserirContrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,(data(Ano,(Inf,Sup),Dia))) :- evolucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,data(Ano,(Inf,Sup),Dia))).
% Contrato - Dia da data
inserirContrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,(data(Ano,Mes,(Inf,Sup)))) :- evolucao(contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,data(Ano,Mes,(Inf,Sup)))).
```

Figura 72: Predicados de inserção de conhecimento impreciso implícito

Remoção de Conhecimento

Quanto à remoção de conhecimento, o formato das operações é uniforme a todos os tipos de conhecimento e bastante simples: o utilizador apenas necessita de indicar o tipo de conhecimento do termo em questão e o respetivo identificador.

Com estes dados, o sistema de inferência encarrega-se de identificar o termo em questão e remove-lo da base de conhecimento, caso exista.

```
% ----- Perfeito ----
% O primeiro argumento deve indicar se o conhecimento e positivo ou negativo

removerAdjudicante(pos,IdAd) :- adjudicanteId(IdAd,Adjudicante), involucaoPos(Adjudicante).
removerAdjudicante(neg,IdAd) :- adjudicanteNegId(IdAd,Lista), involucaoListaNeg(Lista).

removerAdjudicataria(pos,IdAda) :- adjudicatariaId(IdAda,Adjudicataria), involucaoPos(Adjudicataria).
removerAdjudicataria(neg,IdAda) :- adjudicatariaNegId(IdAda,Lista), involucaoListaNeg(Lista).

removerContrato(pos,IdC) :- contratoId(IdC,Contrato), involucaoPos(Contrato).
removerContrato(neg,IdC) :- contratoNegId(IdC,Lista), involucaoListaNeg(Lista).
```

Figura 73: Predicados de remoção de conhecimento perfeito

```
% ----- Imperfeito Incerto ----
% Deve usar as keywords abaixo correspondentes ao parametro incerto que se pretende remover, caso contrario nao funcionara

removerAdjudicante(nomeIncerto,IdAd) :- adjudicanteId(IdAd,Adjudicante), involucao(Adjudicante).
removerAdjudicante(nifIncerto,IdAd) :- adjudicanteId(IdAd,Adjudicante), involucao(Adjudicante).
removerAdjudicante(moradaIncerta,IdAd) :- adjudicanteId(IdAd,Adjudicante), involucao(Adjudicante).

removerAdjudicataria(nomeIncerto,IdAda) :- adjudicatariaId(IdAda,Adjudicataria), involucao(Adjudicataria).
removerAdjudicataria(nifIncerto,IdAda) :- adjudicatariaId(IdAda,Adjudicataria), involucao(Adjudicataria).
removerAdjudicataria(moradaIncerta,IdAda) :- adjudicatariaId(IdAda,Adjudicataria), involucao(Adjudicataria).

removerContrato(idAdIncerto,IdC) :- contratoId(IdC,Contrato), involucao(Contrato).
removerContrato(idAdaIncerto,IdC) :- contratoId(IdC,Contrato), involucao(Contrato).
removerContrato(tipoIncerto,IdC) :- contratoId(IdC,Contrato), involucao(Contrato).
removerContrato(procedimentoIncerto,IdC) :- contratoId(IdC,Contrato), involucao(Contrato).
removerContrato(descricaoIncerta,IdC) :- contratoId(IdC,Contrato), involucao(Contrato).
removerContrato(valorIncerto,IdC) :- contratoId(IdC,Contrato), involucao(Contrato).
removerContrato(prazoIncerto,IdC) :- contratoId(IdC,Contrato), involucao(Contrato).
removerContrato(localIncerto,IdC) :- contratoId(IdC,Contrato), involucao(Contrato).
removerContrato(dataIncerta,IdC) :- contratoId(IdC,Contrato), involucao(Contrato).
```

Figura 74: Predicados de remoção de conhecimento incerto

```
% ----- Imperfeito Interdito ----
% Deve usar as keywords abaixo correspondentes ao parametro interdito que se pretende remover

removerAdjudicante(nomeInterdito,IdAd) :- adjudicanteId(IdAd,Adjudicante), involucao(Adjudicante).
removerAdjudicante(nifInterdito,IdAd) :- adjudicanteId(IdAd,Adjudicante), involucao(Adjudicante).
removerAdjudicante(moradaInterdita,IdAd) :- adjudicanteId(IdAd,Adjudicante), involucao(Adjudicante).

removerAdjudicataria(nomeInterdito,IdAda) :- adjudicatariaId(IdAda,Adjudicataria), involucao(Adjudicataria).
removerAdjudicataria(nifInterdito,IdAda) :- adjudicatariaId(IdAda,Adjudicataria), involucao(Adjudicataria).
removerAdjudicataria(moradaInterdita,IdAda) :- adjudicatariaId(IdAda,Adjudicataria), involucao(Adjudicataria).

removerContrato(idAdInterdito,IdC) :- contratoId(IdC,Contrato), involucao(Contrato).
removerContrato(idAdaInterdito,IdC) :- contratoId(IdC,Contrato), involucao(Contrato).
removerContrato(tipoInterdito,IdC) :- contratoId(IdC,Contrato), involucao(Contrato).
removerContrato(procedimentoInterdito,IdC) :- contratoId(IdC,Contrato), involucao(Contrato).
removerContrato(descricaoInterdita,IdC) :- contratoId(IdC,Contrato), involucao(Contrato).
removerContrato(valorInterdito,IdC) :- contratoId(IdC,Contrato), involucao(Contrato).
removerContrato(prazoInterdito,IdC) :- contratoId(IdC,Contrato), involucao(Contrato).
removerContrato(localInterdito,IdC) :- contratoId(IdC,Contrato), involucao(Contrato).
removerContrato(dataInterdita,IdC) :- contratoId(IdC,Contrato), involucao(Contrato).
```

Figura 75: Predicados de remoção de conhecimento interdito

```
% ----- Imperfeito Impreciso ----

% Predicados completos que constituem conhecimento impreciso
removerAdjudicante(imprecisoExplicito,IdAd) :- adjudicantesImprecisosExplicitosId(IdAd,Lista), involucaoImprecisoExplicito(Lista).
removerAdjudicataria(imprecisoExplicito,IdAda) :- adjudicatariasImprecisasExplicitosId(IdAda,Lista), involucaoImprecisoExplicito(Lista).
removerContrato(imprecisoExplicito,IdC) :- contratosImprecisosExplicitosId(IdC,Lista), involucaoImprecisoExplicito(Lista).

% Predicados que possuem parametros entre certos valores
removerAdjudicante(imprecisoImplicito,IdAd) :- intervaloImprecisoId(adjudicante,IdAd,Intervalo), involucao(Intervalo).
removerAdjudicataria(imprecisoImplicito,IdAda) :- intervaloImprecisoId(adjudicataria,IdAda,Intervalo), involucao(Intervalo).
removerContrato(imprecisoImplicito,IdC) :- intervaloImprecisoId(contrato,IdC,Intervalo), involucao(Intervalo).
```

Figura 76: Predicados de remoção de conhecimento impreciso

3.5.2 Identificações

Para além da evolução e involução de entidades, o grupo elaborou também predicados para a consulta de informação na base de conhecimento, de maneira a tornar o sistema mais prático e operacional. Como tal, o grupo pensou em identificações intuitivas e práticas, que permitissem explorar de forma abrangente a base de conhecimento e fossem úteis de um ponto de vista de utilização.

Para além das identificações, foram elaborados vários predicados auxiliares que permitissem o seu correto funcionamento. Estes podem ser consultados no anexo.

Em primeiro lugar, foram criados predicados para pesquisar um adjudicante, adjudicatária ou contrato por identificador, para qualquer termo não negativo. O processo consiste em utilizador o meta-predicado **solucoes** para criar uma lista com todos os predicados que existam com o id em questão.

Como já foi explicado, não existem predicados não negativos com o mesmo id, logo a lista terá apenas um elemento, pelo que se devolve a cabeça da lista e ignora o resto.

```
% Adjudicante por id
adjudicanteId(IdAd,Adjudicante) :-
    solucoes(adjudicante(IdAd,Nome,NIF,Morada), adjudicante(IdAd,Nome,NIF,Morada), [Adjudicante|_]).

% Adjudicataria por id
adjudicatariaId(IdAda,Adjudicataria) :-
    solucoes(adjudicataria(IdAda,Nome,NIF,Morada), adjudicataria(IdAda,Nome,NIF,Morada), [Adjudicataria|_]).

% Contrato por id
contratoId(IdC,Contrato) :-
    solucoes(contrato(IdC,B,C,D,E,F,G,H,I,J), contrato(IdC,B,C,D,E,F,G,H,I,J), [Contrato|_]).
```

Figura 77: Identificações de conhecimento não negativo por id

Conhecimento Perfeito Positivo

No que toca a este tipo de conhecimento, foi desenvolvidas identificações análogas para os adjudicantes e adjudicatárias, pelo que serão explicadas apenas as dos adjudicantes.

A primeira identificação relativa a este predicado permite listar todos os adjudicantes presentes na base de conhecimento. Através do meta-predicado **solucoes**, é criada uma lista com todos os adjudicantes que obedeçam ao predicado auxiliar **adjudiPerfeito**, que verifica que nenhum dos parâmetros do adjudicante é característico de conhecimento imperfeito (p.e. *nomeIncerto*).

```
% Todos os adjudicantes
adjudicantes(R) :-
    solucoes(adjudicante(IdAd,Nome,NIF,Morada),
            (adjudicante(IdAd,Nome,NIF,Morada), adjudiPerfeito(Nome,NIF,Morada)), R).
```

Figura 78: Identificação de todos os adjudicantes

De seguida, foi criado um predicado para identificar todos os adjudicantes que tenham celebrado algum contrato com uma dada adjudicatária. Neste caso, é criada uma lista com todos os contratos (perfeitos positivos) de uma dada adjudicatária, a partir da qual se cria uma outra lista com todos os adjudicantes que participaram nos contratos em questão, através do predicado auxiliar **adjudicantesDosContratos**. No caso das adjudicatárias, o predicado criado identifica todas as que fecharam contrato com um determinado adjudicante.

```
% Adjudicantes que fecharam contrato com uma dada adjudicataria
adjudicantesContratoComAdjudicataria(IdAda,Adjudicantes) :-
    solucoes(contrato(_,IdAd,IdAda,D,E,F,G,H,I,J),
            (contrato(_,IdAd,IdAda,D,E,F,G,H,I,J), contratoPerfeito(IdAd,IdAda,D,E,F,G,H,I,J)), Contratos),
    adjudicantesDosContratos(Contratos,Adjudicantes).
```

Figura 79: Identificação dos adjudicantes que celebraram algum contrato com uma dada adjudicatária

A terceira identificação criada retorna todos os adjudicantes que possuam contratos ativos, isto é, contratos que, até ao dia de hoje, ainda se encontram em vigor. Começa-se por recorrer ao meta-predicado **solucoes** para criar uma lista com todos os contratos (perfeitos positivos) cujo prazo ainda não tenha expirado, condição que é verificada atráves do predicado auxiliar **prazoExpirado**. De seguida, é criada uma lista com todos os adjudicantes desses contratos, através de **adjudicantesDosContratos**.

```
% Adjudicantes com contratos ativos
adjudicantesContratosAtivos(Adjudicantes) :-
    solucoes(contrato(_,IdAd,C,D,E,F,G,Prazo,I,Data), (contrato(_,IdAd,C,D,E,F,G,Prazo,I,Data),
        contratoPerfeito(IdAd,C,D,E,F,G,Prazo,I,Data), nao(prazoExpirado(Data,Prazo))), Ativos),
    adjudicantesDosContratos(Ativos,Adjudicantes).
```

Figura 80: Identificação dos adjudicantes com contratos ativos

Por fim, foi elaborada uma identificação que calcula os custos totais de um adjudicante. O grupo decidiu restringir este cálculo a contratos concluídos, pois caso contrário não seria correto considerar o valor total do contrato, e a maneira como são distribuídos os pagamentos pode variar de contrato para contrato.

A identificação começa por criar uma lista com os valores de todos os contratos (perfeito positivos) cujo prazo já tenha expirado, com recurso aos predicados auxiliares explicados anteriormente. De seguida, soma todos os valores dessa lista atráves de **valorAcumulado** e devolve o custo total.

No caso das adjudicatárias, o processo é o mesmo, mas convém notar que estão a ser calculadas as **receitas** totais, e não os custos.

```
% Custos totais de todos os contratos terminados de um adjudicante
adjudicanteCustosTotais(IdAd,Custo) :-
    solucoes(Valor, (contrato(_,IdAd,C,D,E,F,Valor,Prazo,I,Data),
            contratoPerfeito(IdAd,C,D,E,F,Valor,Prazo,I,Data), prazoExpirado(Data,Prazo)), ValoresExpirados),
    valorAcumulado(ValoresExpirados,Custo).
```

Figura 81: Identificação dos custos totais de um dado adjudicante

Quanto aos contratos, as identificações elaboradas já são diferentes das que foram explicadas até este ponto.

As primeiras duas identificações servem para listar todos os contratos ativos de um dado adjudicante e adjudicatária, respetivamente. Não se deve confundir isto com o predicado da figura 80, que identifica todos os adjudicantes com contratos ativos.

Contudo, a primeira parte do processo é igual, no que toca ao meta-predicado **solucoes**, com a nuance de que neste caso já se recorre ao predicado auxiliar **contratoPerfeito** para verificar se todos os parâmetros de um contrato estão bem detalhados. Neste caso, devolve-se a lista de contratos criada pelo mesmo.

```
% Contratos ativos de um certo adjudicante
contratosAtivosAdjudicante(IdAd,Ativos) :-
    solucoes(contrato(A,IdAd,C,D,E,F,G,Prazo,I,Data), (contrato(A,IdAd,C,D,E,F,G,Prazo,I,Data),
        contratoPerfeito(IdAd,C,D,E,F,G,Prazo,I,Data), nao(prazoExpirado(Data,Prazo))), Ativos).

% Contratos ativos de uma certa adjudicataria
contratosAtivosAdjudicataria(IdAda,Ativos) :-
    solucoes(contrato(A,B,IdAda,D,E,F,G,Prazo,I,Data), (contrato(A,B,IdAda,D,E,F,G,Prazo,I,Data),
        contratoPerfeito(B,IdAda,D,E,F,G,Prazo,I,Data), nao(prazoExpirado(Data,Prazo))), Ativos).
```

Figura 82: Identificações dos contratos ativos de um certo adjudicante/adjudicatária

A identificação seguinte retorna todos os contratos entre um dado adjudicantes e uma adjudicatária. Recorrendo ao meta-predicado **solucoes**, identifica todos os contratos que possuam os identificadores especificados e que constituam conhecimento perfeito positivo.

```
% Contratos entre um certo adjudicante e uma adjudicataria
contratosEntreAdjudicanteAdjudicataria(IdAd,IdAda,Contratos) :-
    solucoes(contrato(A,IdAd,IdAda,D,E,F,G,H,I,J),
            (contrato(A,IdAd,IdAda,D,E,F,G,H,I,J),contratoPerfeito(IdAd,IdAda,D,E,F,G,H,I,J)), Contratos).
```

Figura 83: Identificação dos contratos entre um adjudicante e uma adjudicatária

Os seguintes predicados seguem todos a mesma linha de raciocínio e listam todos os contratos cujo determinado parâmetro assuma o valor especificado - um certo tipo de contrato, tipo de procedimento, ano ou local de oficialização.

Realizam estas operações procurando todos os contratos (perfeitos positivos) existentes cujo parâmetro em questão possua o valor especificado, com recurso aos predicados auxiliares mencionados acima.

```
% Contratos de um certo tipo
contratosTipoContrato(Tipo,Contratos) :-
    solucoes(contrato(A,B,C,Tipo,E,F,G,H,I,J),
        (contrato(A,B,C,Tipo,E,F,G,H,I,J), contratoPerfeito(B,C,Tipo,E,F,G,H,I,J)), Contratos).

% Contratos com um certo tipo de procedimento
contratosTipoProcedimento(Procedimento,Contratos) :-
    solucoes(contrato(A,B,C,D,Procedimento,F,G,H,I,J),
        (contrato(A,B,C,D,Procedimento,F,G,H,I,J), contratoPerfeito(B,C,D,Procedimento,F,G,H,I,J)), Contratos).

% Contratos celebrados em dado ano
contratosAno(Ano,Contratos) :-
    solucoes(contrato(A,B,C,D,E,F,G,H,I,data(Ano,J,K)),
        (contrato(A,B,C,D,E,F,G,H,I,data(Ano,J,K)), contratoPerfeito(B,C,D,E,F,G,H,I,data(Ano,J,K))), Contratos).

% Contratos fechados num certo local
contratosLocal(Local,Contratos) :-
    solucoes(contrato(A,B,C,D,E,F,G,H,Local,J),
        (contrato(A,B,C,D,E,F,G,H,Local,J), contratoPerfeito(B,C,D,E,F,G,H,Local,J)), Contratos).
```

Figura 84: Identificações dos contratos de acordo com um determinado parâmetro

As últimas duas identificações elaboradas para este tipo de conhecimento devolvem as listas de contratos acima de um determinado valor, ou cujo valor pertença a um dado intervalo, respetivamente.

Através do meta-predicado **solucoes**, vão percorrendo todos os contratos na base de conhecimento, verificando se são conhecimento perfeito positivo e se o respetivo valor obedece às restrições estabelecidas.

```
% Contratos com valor superior a um certo limite
contratosAcimaValor(Limite,Contratos) :-
    solucoes(contrato(A,B,C,D,E,F,Valor,H,I,J), (contrato(A,B,C,D,E,F,Valor,H,I,J),
        contratoPerfeito(B,C,D,E,F,Valor,H,I,J), Valor >= Limite), Contratos).

% Contratos com valor entre dois limites
contratosEntreValores(Inf,Sup,Contratos) :-
    solucoes(contrato(A,B,C,D,E,F,Valor,H,I,J), (contrato(A,B,C,D,E,F,Valor,H,I,J),
        contratoPerfeito(B,C,D,E,F,Valor,H,I,J), Valor >= Inf, Valor =< Sup), Contratos).
```

Figura 85: Identificações dos contratos cujo valor obedeça às restrições estabelecidas

Conhecimento Perfeito Negativo

No que toca a este tipo de conhecimento, as identificações são mais rudimentares, uma vez que conhecimento negativo não é tão relevante como conhecimento positivo, de um ponto de vista realista. Mais uma vez, os predicados de adjudicantes e adjudicatárias são análogos, pelo que desta vez, serão abordados apenas as identificações de adjudicatárias.

A primeira identificação devolve uma lista com todos os factos falsos acerca de uma adjudicatária com um determinado identificador. Recorre-se ao meta-predicado **solucoes** para percorrer todas as adjudicatárias negativas do sistema, e verifica-se ainda se o respetivo campo do NIF está instanciado.

Isto é necessário uma vez que uma das correspondências identificadas terá apenas o identificador instanciado, porque é dado, e os restantes parâmetros serão apontadores. Como não existe tal termo na base de conhecimento, é considerado falso, devido ao **Pressuposto do Mundo Fechado**. Logo, não deve ser considerado.

Depois de terminar a execução do meta-predicado, é necessário ainda verificar que a lista resultante não é vazia. Dado que esta identificação é usada no predicado exibido na figura 68, é necessário que a lista devolvida não seja nula, caso contrário esse predicado daria *true* mesmo que não existisse nenhuma adjudicatária com o id que se pretendesse remover, e o estado da base de conhecimento não se alterasse.

```
% Adjudicatarias negativas por id
adjudicatariaNegId(IdAd,R) :-
    solucoes(-adjudicataria(IdAd,Nome,NIF,Morada), (-adjudicataria(IdAd,Nome,NIF,Morada), number(NIF)), R),
    nao(comprimento(R,0)).
```

Figura 86: Identificação dos termos negativos de uma adjudicatária com um certo id

A segunda identificação lista todos os termos falsos acerca de adjudicatárias, inseridos na base de conhecimento através da **negação forte**.

```
% Adjudicatarias negativas
adjudicatariasNeg(Adjudicatarias) :-
    solucoes(-adjudicataria(IdAd,Nome,NIF,Morada), -adjudicataria(IdAd,Nome,NIF,Morada), Adjudicatarias).
```

Figura 87: Identificação dos termos negativos acerca de adjudicatárias

Por fim, a última identificação devolve uma lista com os NIFs de todos os termos negativos que possuam os restantes parâmetros conforme especificados. Através do meta-predicado **solucoes**, percorre-se todos termos negativos de adjudicatárias existentes, com a salvaguarda relativa à instanciação presente na figura 86, e vai-se guardando os seus NIFs numa lista.

Esta identificação é usada nos invariantes que restringem a inserção de conhecimento impreciso implícito, de maneira a impedir que seja inserido um intervalo cujos valores se saiba que são todos falsos.

```
% NIFS de adjudicatarias negativos com os parametros dados
nifsAdjudicatariasNeg((IdAda,Nome,Morada),NIFS) :-
    solucoes(NIF, (-adjudicataria(IdAda,Nome,NIF,Morada), number(NIF)), NIFS).
```

Figura 88: Identificação dos NIFs dos termos negativos relativos a adjudicatárias

No caso dos contratos, foram criadas identificações análogas às das figuras 86 e 87.

```
% Contratos negativos por id
contratoNegId(IdC,R) :-
    solucoes(-contrato(IdC,B,C,D,E,F,G,H,I,J), (-contrato(IdC,B,C,D,E,F,G,H,I,J), number(G)), R),
    nao(comprimento(R,0)).

% Contratos negativos
contratosNeg(Contratos) :-
    solucoes(-contrato(IdC,B,C,D,E,F,G,H,I,J), -contrato(IdC,B,C,D,E,F,G,H,I,J), Contratos).
```

Figura 89: Identificações de termos negativos relativos a contratos

Para além disso, foram ainda criadas identificações que listam os valores, prazos, anos, meses e dias dos termos negativos de contratos existentes na base de conhecimento, que são análogos ao predicado da figura 88 e usados para o mesmo propósito, que é explicado acima.

```
% Valores de contratos negativos com os parametros dados
valoresContratosNeg((IdC,IdAd,IdAda,TC,TP,Desc,P,L,D),Valores) :-
    solucoes(Valor, (-contrato(IdC,IdAd,IdAda,TC,TP,Desc,Valor,P,L,D), number(Valor)), Valores).

% Prazos de contratos negativos com os parametros dados
prazosContratosNeg((IdC,IdAd,IdAda,TC,TP,Desc,V,L,D),Prazos) :-
    solucoes(Prazo, (-contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,Prazo,L,D), number(Prazo)), Prazos).

% Anos de contratos negativos com os parametros dados
anosContratosNeg((IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,M,D),Anos) :-
    solucoes(Ano, (-contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,data(Ano,M,D)), number(Ano)), Anos).

% Meses de contratos negativos com os parametros dados
mesesContratosNeg((IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,A,D),Meses) :-
    solucoes(Mes, (-contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,data(A,Mes,D)), number(Mes)), Meses).

% Dias de contratos negativos com os parametros dados
diasContratosNeg((IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,A,M),Dias) :-
    solucoes(Dia, (-contrato(IdC,IdAd,IdAda,TC,TP,Desc,V,P,L,data(A,M,Dia)), number(Dia)), Dias).
```

Figura 90: Identificações de termos negativos relativos a contratos

Conhecimento Imperfeito

Relativamente a conhecimento imperfeito incerto e interdito, foram elaboradas identificações para listar todos os termos existentes de um determinado tipo de conhecimento imperfeito, para um dado parâmetro. Isto é, existem predicados para identificar todos os contratos com valor incerto, prazo incerto, local interdito, etc.

No caso de conhecimento imperfeito impreciso, as identificações devolvem todos os termos imprecisos de um predicado com um dado id, no caso de conhecimento **explícito**, ou o termo com o respetivo intervalo de valores, no caso de conhecimento **implícito**.

Não se entrará em grande detalhe nestas identificações porque são bastante simples e fazem exatamente o que foi descrito acima. De seguida, são apresentados alguns exemplos:

```
% Contratos com id de adjudicataria incerto
contratosIdAdaIncerto(R) :-
    solucoes(contrato(A,B,idAdaIncerto,D,E,F,G,H,I,J), contrato(A,B,idAdaIncerto,D,E,F,G,H,I,J), R).

% Contratos de tipo incerto
contratosTipoIncerto(R) :-
    solucoes(contrato(A,B,C,tipoIncerto,E,F,G,H,I,J), contrato(A,B,C,tipoIncerto,E,F,G,H,I,J), R).
```

Figura 91: Identificações de contratos com um certo parâmetro incerto

```
% Todas as possibilidades para o adjudicante com certo id
adjudicantesImprecisosExplicitosId(IdAd,R) :-
    solucoes(adjudicante(IdAd,Nome,NIF,Morada), excecao(adjudicante(IdAd,Nome,NIF,Morada)), R),
    nao(comprimento(R,0)).

% Todas as possibilidades para a adjudicataria com certo id
adjudicatariasImprecisasExplicitasId(IdAda,R) :-
    solucoes(adjudicataria(IdAda,Nome,NIF,Morada), excecao(adjudicataria(IdAda,Nome,NIF,Morada)), R),
    nao(comprimento(R,0)).
```

Figura 92: Identificações de termos imprecisos explícitos

```
% As possibilidades para o adjudicante com um certo id, no formato que o utilizador usa para o introduzir
adjudicantesImprecisosImplicitosId(IdAd,R) :-
    intervaloImprecisoId(adjudicante,IdAd,intervalo(_,_,_,(Inf,Sup))),
    solucoes(adjudicante(IdAd,Nome,(Inf,Sup),Morada), excecao(adjudicante(IdAd,Nome,Inf,Morada)), R).

% As possibilidades para a adjudicataria com um certo id, no formato que o utilizador usa para o introduzir
adjudicatariasImprecisosImplicitosId(IdAda,R) :-
    intervaloImprecisoId(adjudicataria,IdAda,intervalo(_,_,_,(Inf,Sup))),
    solucoes(adjudicataria(IdAda,Nome,(Inf,Sup),Morada), excecao(adjudicataria(IdAda,Nome,Inf,Morada)), R).
```

Figura 93: Identificações de termos imprecisos implícitos

Convém salientar o predicado **intervaloImprecisoId** que, dado o nome do predicado e respetivo id, identifica o intervalo de valores que lhe diz respeito, caso se trate de conhecimento impreciso. É utilizado tanto em algumas identificações como noutras partes do programa, por exemplo, na remoção de conhecimento mostrada na figura 76.

```
% Limites do intervalo de valores imprecisos do termo com certo id
intervaloImprecisoId(Predicado,Id,Intervalo) :-
    solucoes(intervalo(Predicado,Id,Parametro,(Inf,Sup)), intervalo(Predicado,Id,Parametro,(Inf,Sup)), [Intervalo|_]).
```

Figura 94: Identificação do intervalo de valores de um determinado termo

3.6 Extras

De maneira a tornar o sistema de representação de conhecimento e raciocínio o mais completo e prático possível, foram implementados vários recursos extra.

3.6.1 Modularidade de Código

O código do projeto foi dividido ao longo de vários ficheiros e pastas, de maneira a torna-lo o mais modular possível e simples de encontrar. Será então explicada a estrutura do trabalho, que é possível observar de seguida:

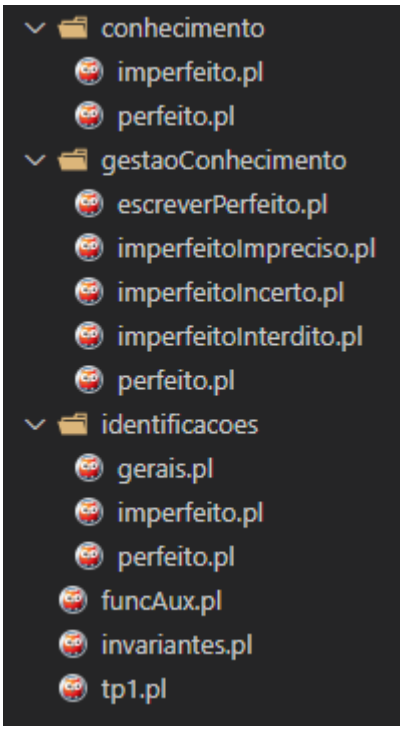

Figura 95: Estrutura do projeto

Existe um ficheiro **funcAux** que possui todos os predicados auxiliares desenvolvidos e um ficheiro **invariantes**, cujo nome é bastante sugestivo.

Além disso, existe uma pasta **identificacoes**, onde estão guardadas todas as identificações do programa, distribuídas por três ficheiros distintos: **gerais.pl** (identificações válidas para qualquer conhecimento não negativo), **perfeito.pl** (conhecimento perfeito) e **imperfeito.pl** (conhecimento imperfeito).

Na pasta **conhecimento**, encontram-se os ficheiros responsáveis pelo povoamento da base de conhecimento: **perfeito.pl** (conhecimento perfeito) e **imperfeito.pl** (conhecimento imperfeito).

A pasta **gestaoConhecimento** alberga os ficheiros responsáveis pela evolução e involução dos vários tipos de conhecimento, bem como o ficheiro **escreverPerfeito.pl**, outro extra implementado pelo grupo que será abordado em 3.6.2.

Por fim, o ficheiro principal é o **tp1.pl**, que inclui todos os restantes ficheiros e módulos necessários (para obter a data atual), como pode ser observado na figura 96, encarrega-se das declarações iniciais necessárias ao correto funcionamento do programa (figura 97) e dispõe das operações simplificadas de inserção e remoção de conhecimento explicadas em 3.5.1. O programa deve ser consultado/compilado a partir deste ficheiro.

```
%------------------------------ - - - - - - - - - - - -
% Carregar predicados de ficheiros auxiliares
:- include('invariantes.pl').
:- include('funcAux.pl').

:- include('identificacoes/gerais.pl').
:- include('identificacoes/perfeito.pl').
:- include('identificacoes/imperfeito.pl').

:- include('gestaoConhecimento/perfeito.pl').
:- include('gestaoConhecimento/escreverPerfeito.pl').
:- include('gestaoConhecimento/imperfeitoImpreciso.pl').
:- include('gestaoConhecimento/imperfeitoIncerto.pl').
:- include('gestaoConhecimento/imperfeitoInterdito.pl').

:- include('conhecimento/perfeito.pl').
:- include('conhecimento/imperfeito.pl').

%------------------------------ - - - - - - - - - - - -
% Modulo para obter a data atual
:- use_module(library(system)).
```

Figura 96: Inclusão de ficheiros e módulos auxiliares

```
%------------------------------ - - - - - - - - -
% SICStus PROLOG: Declaracoes iniciais

:- set_prolog_flag( discontiguous_warnings,off ).
:- set_prolog_flag( single_var_warnings,off ).
:- set_prolog_flag( unknown,fail ).

:- op(900,xfy,'::').

:- dynamic (::)/2.
:- dynamic '-'/1.
:- dynamic adjudicante/4.
:- dynamic adjudicataria/4.
:- dynamic contrato/10.
:- dynamic intervalo/4.
:- dynamic excecao/1.
:- dynamic nuloInterdito/1.
```

Figura 97: Declarações iniciais

3.6.2 Persistência de Informação

É possível introduzir e remover informação da base de conhecimento através dos predicados de evolução e involução criados, contudo estes predicados não asseguram a sua persistência. Ou seja, toda a informação alterada em tempo de execução é perdida quando o programa é interrompido.

Tendo isto em mente, o grupo decidiu desenvolver o predicado **escreverPerfeito** para guardar o estado da base de conhecimento perfeito do programa e, assim, garantir persistência dessa mesma informação.

Para esse efeito, faz-se recurso a três predicados do *PROLOG* - **open**, **write** e **close**. Com o **open**, abre-se, em modo de escrita, o ficheiro no qual será guardada a informação - **conhecimento/perfeito.pl** - e a *Stream* de *input/output*. O **write** escreve para a *Stream* o conhecimento em questão e respetivas legendas. Por último, o **close** fecha a *Stream*, sendo todo o seu conteúdo efetivamente escrito no ficheiro.

```
escreverPerfeito :-
    open('conhecimento/perfeito.pl', write, Stream),

    write(Stream, '%------------------------------ - - - - - - - - - - - - - - - -\n'),
    write(Stream, '% Conhecimento perfeito positivo\n'),
    write(Stream, '\n% Adjudicante: #IdAd, Nome, NIF, Morada\n'),
    writeAdjudicantePos(Stream),
    write(Stream, '\n% Adjudicataria: #IdAda, Nome, NIF, Morada\n'),
    writeAdjudicatariaPos(Stream),
    write(Stream, '\n% Contrato: #IdC, #IdAd, #IdAda, Tipo, Procedimento, Descricao, Valor, Prazo, Local, Data\n'),
    writeContratoPos(Stream),

    write(Stream, '\n\n%------------------------------ - - - - - - - - - - - - - - - -\n'),
    write(Stream, '% Conhecimento perfeito negativo\n'),
    write(Stream, '\n% Adjudicante: #IdAd, Nome, NIF, Morada\n'),
    writeAdjudicanteNeg(Stream),
    write(Stream, '\n% Adjudicataria: #IdAda, Nome, NIF, Morada\n'),
    writeAdjudicatariaNeg(Stream),
    write(Stream, '\n% Contrato: #IdC, #IdAd, #IdAda, Tipo, Procedimento, Descricao, Valor, Prazo, Local, Data\n'),
    writeContratoNeg(Stream),

    close(Stream).
```

Figura 98: Predicado **escreverPerfeito**

Para cada tipo de conhecimento perfeito foi desenvolvido um predicado responsável por escrever o termo para a *Stream*, da maneira que é mais conveniente:

```
writeAdjudicantePos(Stream) :- adjudicante(IdAd,Nome,NIF,Morada),
        adjudiPerfeito(Nome,NIF,Morada),
        write(Stream, 'adjudicante('), write(Stream, IdAd), write(Stream, ',\''),
        write(Stream, Nome), write(Stream, '\','),
        write(Stream, NIF), write(Stream, ',\''),
        write(Stream, Morada), write(Stream, '\').\n'),
    fail; true.

writeAdjudicanteNeg(Stream) :- -adjudicante(IdAd,Nome,NIF,Morada),
        write(Stream, '-adjudicante('), write(Stream, IdAd), write(Stream, ',\''),
        write(Stream, Nome), write(Stream, '\','),
        write(Stream, NIF), write(Stream, ',\''),
        write(Stream, Morada), write(Stream, '\').\n'),
    fail; true.
```

Figura 99: Predicados de escrita de adjudicantes

```
writeAdjudicatariaPos(Stream) :- adjudicataria(IdAda,Nome,NIF,Morada),
        adjudiPerfeito(Nome,NIF,Morada),
        write(Stream, 'adjudicataria('), write(Stream, IdAda), write(Stream, ',\''),
        write(Stream, Nome), write(Stream, '\','),
        write(Stream, NIF), write(Stream, ',\''),
        write(Stream, Morada), write(Stream, '\').\n'),
    fail; true.

writeAdjudicatariaNeg(Stream) :- -adjudicataria(IdAda,Nome,NIF,Morada),
        write(Stream, '-adjudicataria('), write(Stream, IdAda), write(Stream, ',\''),
        write(Stream, Nome), write(Stream, '\','),
        write(Stream, NIF), write(Stream, ',\''),
        write(Stream, Morada), write(Stream, '\').\n'),
    fail; true.
```

Figura 100: Predicados de escrita de adjudicatárias

```
writeContratoPos(Stream) :-
    contrato(IdC,IdAd,IdAda,TipoContrato,TipoProcedimento,Descricao,Valor,Prazo,Local,Data),
        contratoPerfeito(IdAd,IdAda,TipoContrato,TipoProcedimento,Descricao,Valor,Prazo,Local,Data),
        write(Stream, 'contrato('), write(Stream, IdC), write(Stream, ','),
        write(Stream, IdAd), write(Stream, ','),
        write(Stream, IdAda), write(Stream, ',\''),
        write(Stream, TipoContrato), write(Stream, '\',\''),
        write(Stream, TipoProcedimento), write(Stream, '\',\''),
        write(Stream, Descricao), write(Stream, '\','),
        write(Stream, Valor), write(Stream, ','),
        write(Stream, Prazo), write(Stream, ',\''),
        write(Stream, Local), write(Stream, '\','),
        write(Stream, Data), write(Stream, ').\n'),
    fail; true.

writeContratoNeg(Stream) :-
    -contrato(IdC,IdAd,IdAda,TipoContrato,TipoProcedimento,Descricao,Valor,Prazo,Local,Data),
        write(Stream, '-contrato('), write(Stream, IdC), write(Stream, ','),
        write(Stream, IdAd), write(Stream, ','),
        write(Stream, IdAda), write(Stream, ',\''),
        write(Stream, TipoContrato), write(Stream, '\',\''),
        write(Stream, TipoProcedimento), write(Stream, '\',\''),
        write(Stream, Descricao), write(Stream, '\','),
        write(Stream, Valor), write(Stream, ','),
        write(Stream, Prazo), write(Stream, ',\''),
        write(Stream, Local), write(Stream, '\','),
        write(Stream, Data), write(Stream, ').\n'),
        fail; true.
```

Figura 101: Predicados de escrita de contratos

Para recorrer a este predicado, o utilizador deve mudar a sua *Working Directory* para a pasta de onde compilou o ficheiro **tp1.pl**, dentro do programa *SICStus*. Este programa carrega o código para a sua diretoria de instalação, pelo que é necessário mudar a diretoria de execução manualmente para a do projeto, de maneira a que o programa reconheça a referência ao ficheiro **conhecimento/perfeito.pl** existente no código.

O grupo ponderou em fazer um predicado equivalente para guardar o estado da base de conhecimento imperfeito, contudo decidiu não avançar com isto devido à maneira como foi realizado o povoamento inicial: cada termo imperfeito encontra-se comentado, explicando o seu significado, e o predicado de escrita sobreporia os comentários. Também foi ponderada a escrita num ficheiro diferente, mas isso teria problemas associados, uma vez que se ficaria com informação repetida, dado que os termos do povoamento inicial estariam em ambos os ficheiros.

Contudo, fica a salvaguarda de que criar tal predicado não seria difícil, mas também não é muito importante para o presente projeto.

4 Conclusões e Sugestões

O desenvolvimento deste projeto permitiu a consolidação da matéria teórico-prática lecionada nas aulas da unidade curricular e a eludicidação da verdadeira utilidade e poder de ferramentas como a linguagem *PROLOG* e o programa *SICStus*, no que toca à construção de bases de conhecimento dinâmicas e sistemas de inferência complexos, para o tratamento de conhecimento e modulação de cenários reais.

O grupo conclui está satisfeito com o resultado final e considera que desenvolveu com sucesso um sistema de representação de conhecimento e raciocínio que caracteriza bem a área dos Contratos Públicos e das normas de adjudicação portuguesa, tendo cumprido os requisitos propostos no enunciado, além de ainda implementar algumas ferramentas extra, de maneira a tornar o ambiente mais robusto, prático e fácil de utilizar.

O sistema criado permite a representação de todo o tipo de conhecimento estudado, tanto perfeito (positivo e negativo) como imperfeito (incerto, impreciso e interdito), além de facultar, por um lado, vários exemplos e demonstrações dos tipos de conhecimento e, por outro, ferramentas para o próprio utilizador manipular o conhecimento ao seu gosto - predicados de evolução e involução -, bem como predicados simplificados para a realização destas operações.

Foram elaborados inúmeros invariantes, com o intuito de garantir a consistência e veracidade da informação armazenada, evitando assim conflitos entre os diferentes tipos de conhecimento e o manuseamento de informação inválida, de um ponto de vista real. Além disso, o sistema de inferência implementado permite a operação em programação a lógica extendida e a representação de conhecimento imperfeito.

Em futuras iterações, seria possível expandir o nível de detalhe do sistema, entrando em maior pormenor na área de Contratos Públicos, bem como elaborar mais e mais interessantes identificações, em função da nova informação abrangida, e implementar um método de persistência de dados imperfeitos.

O esforço coletivo do grupo e a pesquisa extensiva do caso de estudo, motor de inferência e do paradigma de programação em lógica possibilitou o desenvolvimento controlado e ritmado do projeto.

5 Referências

5.1 Referências Bibliográficas

[Analide, 2011] ANALIDE, Cesar, NOVAIS, Paulo, NEVES, José,
"Sugestões para a Redacção de Relatórios Técnicos",
Relatório Técnico, Departamento de Informática, Universidade do Minho,
Portugal, 2011

[Analide, 1996] ANALIDE, Cesar, NEVES, José,
"Representação de Informação Incompleta",
Texto Pedagógico, Grupo de Inteligência Artificial, Centro de Ciências e
Tecnologias da Computação, Portugal, 1996

5.2 Referências Eletrónicas

BASE: Contratos Públicos Online - Perguntas Frequentes
Disponível em: www.base.gov.pt/Base/pt/PerguntasFrequentes
Acesso em: 02/04/2020

PROLOG
Disponível em: https://pt.wikipedia.org/wiki/Prolog
Acesso em: 04/04/2020

SICStus Prolog
Disponível em: https://sicstus.sics.se/sicstus/docs/latest4/html/sicstus.html/
Acesso em: 13/04/2020

A Anexo

Tal como já foi referido, foram desenvolvidos vários predicados auxiliares, alguns dos quais foram sendo referenciados ao longo do presente relátorio, com o intuito de possibilitar as operações necessárias para o cumprimento dos objetivos do trabalho prático.

A.1 Meta-Predicados

Os meta-predicados **insercao** e **remocao** foram essenciais para o processo de evolução e involução de conhecimento. Numa primeira fase, dão **assert** ou **retract**, respetivamente, do novo termo na base de conhecimento. Se a sua execução não terminar por aí (no contexto do programa, se os termos não obecederem a todos os respetivos invariantes), revertem a operação anterior, retornando a base de conhecimento ao estado original.

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

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

Figura 102: Meta-predicados **insercao** e **remocao**

O meta-predicado **teste** recebe uma lista de predicados e verifica se são todos verdadeiros. Foi usado para verificar os invariantes aquando da evolução/involução de conhecimento.

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

Figura 103: Meta-predicado **teste**

O **demo** permitiu expandir o sistema à programação em lógica extendida, tomando o papel de predicado de inferência com três valores distintos: verdadeiro (se o termo existir na base de conhecimento), falso (caso o termo exista por negação forte) e desconhecido (se nenhuma das condições anteriores se verificar).

```
% Testar conhecimento {V,F,D}
demo(Questao,verdadeiro) :- Questao.
demo(Questao,falso) :- -Questao.
demo(Questao,desconhecido) :- nao(Questao), nao(-Questao).
```

Figura 104: Meta-predicado **demo**

O meta-predicado **nao** retorna o valor lógico contrário ao do predicado que recebe como argumento. Isto é, se receber um predicado verdadeiro, devolve falso, e vice-versa. Foi usado em várias ocasiões, na verificação de condições.

```
% Extensao do meta-predicado nao: Questao -> {V,F}
nao( Questao ) :- Questao, !, fail.
nao( _ ).
```

Figura 105: Meta-predicado **nao**

O **solucoes** cria uma lista Z com as soluções X para todos o Y que encontrar. Foi fundamental para a parte das identificações e para alguns predicados auxiliares.

```
solucoes( X,Y,Z ) :- findall( X,Y,Z ).
```

Figura 106: Meta-predicado **solucoes**

A.2 Predicados Auxiliares

A.2.1 Listas

Em primeiro lugar, foram desenvolvidos vários predicados para efetuar operações genéricas sobre listas.

O predicado auxiliar **comprimento** calcula, tal como o nome sugere, o comprimento de uma lista.

```
% Comprimento de uma lista
comprimento(S,N) :- length(S,N).
```

Figura 107: Predicado auxiliar **comprimento**

O predicado **pertence** verifica se um dado elemento pertence a uma lista.

```
% Verifica se um elemento pertence a uma lista
pertence(X,[X|_]).
pertence(X,[_|T]) :- pertence(X,T).
```

Figura 108: Predicado auxiliar **pertence**

O predicado auxiliar **diferentes** conta o número de elementos diferentes de uma lista.

```
% Conta o numero de elementos diferentes numa lista
diferentes([],0).
diferentes([H|T],N) :- pertence(H,T), diferentes(T,N).
diferentes([H|T],N) :- diferentes(T,N1), N is N1+1.
```

Figura 109: Predicado auxiliar **diferentes**

eliminaRepetidos cria uma lista a partir de outra, retirando todos os elementos repetidos da primeira, com recurso ao predicado **eliminaRepAux**.

```
% Elimina os elementos repetidos numa lista
eliminaRepetidos(L,R) :- eliminaRepAux(L,[],R).

eliminaRepAux([],Temp,Temp).
eliminaRepAux([H|T],Temp,R) :- pertence(H,Temp), eliminaRepAux(T,Temp,R).
eliminaRepAux([H|T],Temp,R) :- eliminaRepAux(T,[H|Temp],R).
```

Figura 110: Predicado auxiliar **eliminaRepetidos**

A.2.2 Datas

O grupo criou também um conjunto de predicados para trabalhar com datas, dado que eram um elemento fulcral dos contratos.

O predicado auxiliar **elementoData** devolve o elemento pretendido - ano, mês ou dia - de um predicado do tipo *data(Ano,Mes,Dia)*.

```
% Extrai um elemento da data
elementoData(data(A,_,_),ano,A).
elementoData(data(_,M,_),mes,M).
elementoData(data(_,_,D),dia,D).
```

Figura 111: Predicado auxiliar **elementoData**

Foi necessário criar o predicado **calculaPrazo** para descobrir a data de expiração dos con-

tratos, uma vez que a data de efetivação dos contratos se encontra no formato *data(Ano,Mes,Dia)* e o prazo é um inteiro.

```
% Calcula o prazo de um contrato
calculaPrazo(A,M,D,0,data(A,M,D)).

calculaPrazo(A,M,D,P,R) :-
    pertence(M,[1,3,5,7,8,10]), NovoPrazo is P-(31-D+1), NovoPrazo >= 0, NovoMes is M+1, calculaPrazo(A,NovoMes,1, NovoPrazo, R).
calculaPrazo(A,M,D,P,R) :- pertence(M,[1,3,5,7,8,10]), NovoDia is D+P, calculaPrazo(A,M,NovoDia, 0, R).

calculaPrazo(A,M,D,P,R) :-
    pertence(M,[4,6,9,11]), NovoPrazo is P-(30-D+1), NovoPrazo >= 0, NovoMes is M+1, calculaPrazo(A,NovoMes,1, NovoPrazo, R).
calculaPrazo(A,M,D,P,R) :- pertence(M,[4,6,9,11]), NovoDia is D+P, calculaPrazo(A,M,NovoDia, 0, R).

calculaPrazo(A,M,D,P,R) :-
    M =:= 12, NovoPrazo is P-(31-D+1), NovoPrazo >= 0, NovoAno is A+1, calculaPrazo(NovoAno,1,1, NovoPrazo, R).
calculaPrazo(A,M,D,P,R) :- M =:= 12, NovoDia is D+P, calculaPrazo(A,M,NovoDia, 0, R).

calculaPrazo(A,M,D,P,R) :-
    M =:= 2, mod(A,4) =:= 0, NovoPrazo is P-(29-D+1), NovoPrazo >= 0, calculaPrazo(A,3,1, NovoPrazo, R).
calculaPrazo(A,M,D,P,R) :- M =:= 2, mod(A,4) =:= 0, NovoDia is D+P, calculaPrazo(A,M,NovoDia, 0, R).

calculaPrazo(A,M,D,P,R) :-
    M =:= 2, NovoPrazo is P-(28-D+1), NovoPrazo >= 0, calculaPrazo(A,3,1, NovoPrazo, R).
calculaPrazo(A,M,D,P,R) :- M =:= 2, NovoDia is D+P, calculaPrazo(A,M,NovoDia, 0, R).
```

Figura 112: Predicado auxiliar **calculaPrazo**

O predicado **comparaDatas** verifica se a segunda data que recebe é cronologicamente anterior à primeira, e foi usado para saber se os contratos ainda se encontravam vigentes ou não.

```
% Verifica se a segunda data e anterior a primeira
comparaDatas(datime(AH,MH,DH,_,_,_),data(AP,MP,DP)) :-
    AP < AH;
    (AP =:= AH, (MP < MH;
                 (MP =:= MH, DP < DH))).
```

Figura 113: Predicado auxiliar **comparaDatas**

O grupo criou o predicado auxiliar **dataValida** para verificar se as datas introduzidas aquando da inserção de novos contratos na base de conhecimento eram válidas ou não - por exemplo, não aceita a data 30/02/2020, uma vez que o mês de fevereiro nunca tem 30 dias.

```
% Verifica se uma data e valida
dataValida(data(A,M,D)) :-
    (pertence(M,[1,3,5,7,8,10,12]), D >= 1, D =< 31);
    (pertence(M,[4,6,9,11]), D >= 1, D =< 30);
    (M =:= 2, (mod(A,4) =:= 0, D >= 1, D =< 29);
                               (D >= 1, D =< 28)).
```

Figura 114: Predicado auxiliar **dataValida**

Por fim, o predicado **prazoExpirado** indica se a validade de um determinado contratos já acabou ou não, recorrendo a alguns predicados auxiliares demonstrados acima e ao meta-predicado **datime** que indica o dia e hora atuais.

```
% Verifica se um contrato ainda esta em vigor
prazoExpirado(data(A,M,D), P) :-
    calculaPrazo(A,M,D,P,Prazo),
    datime(Hoje), !,
    comparaDatas(Hoje,Prazo).
```

Figura 115: Predicado auxiliar **prazoExpirado**

A.2.3 Invariantes

Os predicados auxiliares explanados nesta secção foram elaborados para realizar operações necessárias para os invariantes.

Foram criados predicados relativos a parâmetros do sistema passíveis de serem incertos ou interditos, cujo nome segue o formato *[nomeParametro]Imperfeito*, que verificam se um dado parâmetro é desconhecido ou não. Serão exibidos apenas dois exemplos, dado que os restantes são análogos.

```
% Verifica se o NIF dado pertence a um predicado de conhecimento perfeito
nifImperfeito(NIF) :- NIF = nifIncerto; NIF = nifInterdito.

% Verifica se o id de adjudicante dado pertence a um predicado de conhecimento perfeito
idAdImperfeito(IdAd) :- IdAd = idAdIncerto; IdAd = idAdInterdito.
```

Figura 116: Predicados auxiliares **nifImperfeito** e **idAdImperfeito**

Pela mesma linha de raciocínio, foram criados predicados que verificam de uma só vez se um certo predicado (adjudicante, adjudicatária ou contrato) constitui conhecimento imperfeito, recebendo todos os seus argumento passíveis de serem desconhecidos e fazendo as verificações mencionadas acima. O predicado relativo a contratos é análogo ao seguinte, que é aplicável a adjudicantes e ajudicatárias:

```
adjudiPerfeito(Nome,NIF,Morada) :-
    Nome \= nomeIncerto, Nome \= nomeInterdito,
    NIF \= nifIncerto, NIF \= nifInterdito,
    Morada \= moradaIncerta, Morada \= moradaInterdita.
```

Figura 117: Predicado auxiliar **adjudiImperfeito**

O grupo desenvolveu também predicados que, partindo dos parâmetros de um dado termo,

verificam se já existe uma exceção relativo ao mesmo na base de conhecimento. São usados na inserção de conhecimento imperfeito. A lógica dos contratos é semelhante à seguinte:

```
existeExcecaoAdjudi(IdAd,N,NIF,M) :-
    ((N = nomeIncerto; N = nomeInterdito), demo(adjudicante(IdAd,teste,NIF,M), desconhecido));
    ((NIF = nifIncerto; NIF = nifIncerto), demo(adjudicante(IdAd,N,teste,M), desconhecido));
    ((M = moradaIncerta; M = moradaIncerta), demo(adjudicante(IdAd,N,NIF,teste), desconhecido)).
```

Figura 118: Predicado auxiliar **existeExcecaoAdjudi**

Além disto, foram criados predicados auxiliares para verificar se um dado átomo/objeto é um NIF válido, um número não negativo, um procedimento válido ou um tipo de contrato válido para contratos por ajuste direto, respetivamente.

```
% NIF valido
nifValido(NIF) :- number(NIF), NIF >= 100000000, NIF =< 999999999.

% Numero >= 0
numeroValido(N) :- number(N), N >= 0.

% Tipo de procedimento valido
tipoProcedimento('Ajuste Direto').
tipoProcedimento('Consulta Previa').
tipoProcedimento('Concurso Publico').

% Tipo de contrato valido por ajuste direto
tipoContratoAjusteDireto('Aquisicao').
tipoContratoAjusteDireto('Locacao de Bens Moveis').
tipoContratoAjusteDireto('Aquisicao de Servicos').
```

Figura 119: Predicados auxiliares de validação de parâmetros

O predicado auxiliar **valorAcumulado** soma os valores de uma lista, e o predicado **entreLimites** conta o número de elementos de uma lista que se encontrão entre certos limites.

```
% Valor acumulado dos contratos celebrados no ano da data D ou nos dois anteriores
valorAcumulado([],0).
valorAcumulado([V|T],R) :- valorAcumulado(T,R1), R is R1+V.

% Verifica quantos valores do intervalo estao entre os limites dados
entreLimites([],_,_,0).
entreLimites([X|T], Inf, Sup, N) :-
    X >= Inf, X =< Sup, entreLimites(T,Inf,Sup,N1), N is N1+1.
entreLimites([_|T], Inf, Sup, N) :-
    entreLimites(T,Inf,Sup,N).
```

Figura 120: Predicados auxiliares **valorAcumulado** e **entreLimites**

Por fim, o grupo desenvolveu um predicado auxiliar para garantir que nem todos os valores de um dado termo pertencentes a um certo intervalo são falsos, de maneira a impossibilitar a inserção de conhecimento imperfeito implícito falso. Existe uma instância deste predicado para cada parâmetro inteiro passível de ser desconhecido, pelo que serão exibidos apenas uns exemplos.

```
intervaloNaoTodoNegativo(adjudicataria(IdAda,N,(Inf,Sup),M)) :-
    nifsAdjudicatariasNeg((IdAda,N,M),NIFS),
    Maximo is Sup-Inf+1,
    nao(entreLimites(NIFS,Inf,Sup,Maximo)).

intervaloNaoTodoNegativo(contrato(IdC,IdAd,IdAda,TC,TP,Desc,(Inf,Sup),P,L,D)) :-
    valoresContratosNeg((IdC,IdAd,IdAda,TC,TP,Desc,P,L,D),Valores),
    Maximo is Sup-Inf+1,
    nao(entreLimites(Valores,Inf,Sup,Maximo)).
```

Figura 121: Predicado auxiliar **intervaloNaoTodoNegativo**

A.2.4 Identificações

Em último lugar, foram criados alguns predicados auxiliares às identificações. Existe um predicado **listaAdjudicantesComRepetidos**, que recebe uma lista de contratos e cria outra lista com todos os adjudicantes presentes nesses mesmos contratos, não discriminando elementos repetidos.

```
% Adiciona adjudicante de cada contrato a lista resultado - pode ter repetidos
listaAdjudicantesComRepetidos([],[]).
listaAdjudicantesComRepetidos([contrato(_,IdAd,_,_,_,_,_,_,_,_)|T],[Adjudicante|R]) :-
    adjudicanteId(IdAd,Adjudicante),
    listaAdjudicantesComRepetidos(T,R).
```

Figura 122: Predicado auxiliar **listaAdjudicantesComRepetidos**

O predicado auxiliar **adjudicantesDosContratos** recorre ao predicado supracitado e ao **eliminaRepetidos** para elaborar uma lista de todos os adjudicantes que participaram em dados contratos, sem ocorrências repetidas.

```
% Retorna lista dos adjudicantes que participaram nos contratos em questao, sem repetidos
adjudicantesDosContratos(Contratos,Adjudicantes) :-
    listaAdjudicantesComRepetidos(Contratos,ListaRepetidos),
    eliminaRepetidos(ListaRepetidos,Adjudicantes).
```

Figura 123: Predicado auxiliar **adjudicantesDosContratos**

Exitem predicados auxiliares análogos aos últimos dois demonstrados, aplicados a adjudicatárias.