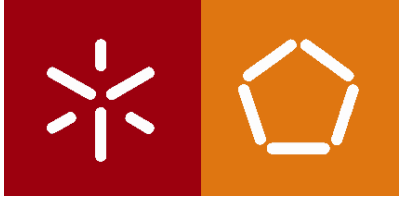


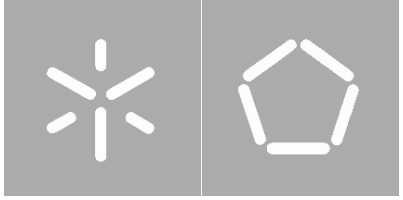
Universidade do Minho

Departamento de Informática

Representação de Informação Incompleta

Cesar Analide

José Neves



Universidade do Minho

Departamento de Informática

Representação de Informação Incompleta

Cesar Analide

José Neves

Texto Pedagógico

Grupo de Inteligência Artificial

Centro de Ciências e Tecnologias da Computação

Índice

1 Representação de Informação Incompleta

Qualquer sistema computacional necessita de armazenar e manipular informação. Os sistemas de Bases de Dados (BD's) e os sistemas de representação de conhecimento, lidam com aspectos concretos do mundo real e podem-se comparar nos termos em que dividem a utilização da informação. Ambos os sistemas distinguem as funções de ***representação*** – descrição do esquema conceptual e dos dados – e de ***computação*** – construção de respostas a questões e manipulação dos dados.

Assim, os requisitos das BD's e dos sistemas de representação de conhecimento para o tratamento da informação são diferentes, como caracterizou *Ullrich Hustadt* [Hus94], questionando a necessidade da adopção do Pressuposto do Mundo Fechado (PMF) em sistemas de representação de conhecimento.

As linguagens de manipulação de informação num sistema de BD's alicerçam-se, basicamente, nos seguintes pressupostos:

Pressuposto do Mundo Fechado – toda a informação que não existe mencionada na base de dados é considerada falsa;

Pressuposto dos Nomes Únicos – duas constantes diferentes (que definam valores atómicos ou objectos) designam, necessariamente, duas entidades diferentes do universo de discurso;

Pressuposto do Domínio Fechado – não existem mais objectos no universo de discurso para além daqueles designados por constantes na base de dados.

Um dos modelos mais importantes, no que concerne à manipulação de informação de BD's, é o ***modelo relacional***, que descreve a informação em termos

de valores atómicos e relações entre conjuntos desses valores.

Contudo, a tipificação da informação de um sistema de representação de conhecimento não se pode fazer nos mesmos termos em que se caracteriza uma BD convencional. Nem sempre se pretende assumir que a informação representada é a única que é válida, e, muito menos, que as entidades representadas sejam as únicas existentes no mundo exterior. Os pressupostos em que se pretende basear um sistema de representação de conhecimento, passam por:

Pressuposto do Mundo Aberto – podem existir outros factos ou conclusões verdadeiros para além daqueles representados na base de conhecimento;

Pressuposto dos Nomes Únicos – duas constantes diferentes (que definam valores atómicos ou objectos) designam, necessariamente, duas entidades diferentes do universo de discurso;

Pressuposto do Domínio Aberto – podem existir mais objectos do universo de discurso para além daqueles designados pelas constantes da base de conhecimento.

Todo este trabalho vai no sentido de dotar um sistema de representação de conhecimento de características capazes de o fazer raciocinar segundo estes três pressupostos.

A implementação de esquemas de raciocínio não-monótono, abordado em [Ana96], proporcionam uma maior flexibilidade no mecanismo de inferência e no processo de obtenção de conclusões, mas fazem-no recorrendo à negação por falha, adoptando o PMF. Um sistema capaz de completar raciocínios não-monótonos não significa, por si só, que possa tratar informação incompleta.

As linguagens tradicionais de Programação em Lógica (PL) proporcionam uma ferramenta poderosa para a representação de conhecimento, uma vez que a característica não-monótona da negação por falha torna possível expressar várias

situações de senso comum que não são disponibilizadas pela lógica clássica. Contudo, a PL tradicional não permite representar directamente informação incompleta. Um sistema de raciocínio sobre uma base de conhecimento com informação incompleta, contempla questões de representação e de inferência diferentes das que se podem identificar num sistema não-monótono, função do tipo de respostas que se pretendem obter em cada um dos sistemas.

Apesar de, numa teoria lógica clássica, o conhecimento ser equacionado em termos do que se prova ser ***verdadeiro***, o que se prova ser ***falso*** e o que representa informação sobre a qual não se pode ser conclusivo, num programa em lógica, as respostas às questões colocadas são, apenas, de dois tipos: ***verdadeiro*** ou ***falso***. Isto deve-se ao facto de um programa em lógica apresentar várias limitações em termos da representação de conhecimento (não permite representar explicitamente informação negativa ou disjuntiva), para além do facto de que, em termos de uma semântica operacional se aplicar, automaticamente, o PMF a todos os predicados.

A generalidade dos programas escritos em lógica representa implicitamente a informação negativa, assumindo a aplicação do raciocínio segundo o PMF. Uma extensão de um programa em lógica pode incluir informação negativa explicitamente, bem como explicitar directamente o PMF para alguns predicados. Consequentemente, torna-se possível distinguir três tipos de conclusões para uma questão: esta pode ser ***verdadeira***, ***falsa*** ou, quando não existe informação que permita inferir uma ou outra das conclusões anteriores, a resposta à questão será ***desconhecida***.

2 Extensão à Programação em Lógica

Tal como tem ficado marcado pela discussão que foi realizada, um programa em lógica determina respostas em termos da veracidade ou falsidade das questões, não sendo possível, assim, implementar um mecanismo de raciocínio que possibilite

abordar a representação de informação incompleta.

De uma forma bastante simples, pode-se descrever através de um programa em lógica, a asserção segundo a qual "*as aves voam*", representando-a como:

$$voa(X) \leftarrow ave(X)$$

De forma idêntica, é possível representar a declaração de que "*as avestruzes não voam*", pela introdução de um segundo predicado, $não\text{-}voa(X)$, cujo significado é a antítese do predicado $voa(X)$, e declarar:

$$não\text{-}voa(X) \leftarrow avestruz(X)$$

ou seja, o elemento *X* ***não voa*** se for uma avestruz. Contudo, esta representação não capta a relação que deve existir entre a informação representada por cada um dos predicados. Isto porque, computacionalmente, se tratam de dois predicados diferentes, sem qualquer tipo de relação definida entre si. Intuitivamente, é óbvio que essa relação deveria existir.

A abordagem de problemas em PL seria muito mais natural se fosse possível declarar explicitamente informação falsa (negativa), já que este tipo de informação tem um papel influente no raciocínio por senso comum.

O objectivo de estender a PL é o de que passe a permitir representar informação negativa explicitamente. A extensão de um programa em lógica passa a contar com dois tipos de negação: a ***negação por falha***, característica dos programas em lógica tradicionais, representada pelo termo $não$, e a ***negação forte*** ou ***clássica***, como forma de identificar informação negativa, ou falsa, representada pela conectiva '¬'.

Deste modo, a representação da afirmação que dava conta que "*as avestruzes não voam*" seria descrita pela negação, explícita, do significado atribuído ao predicado $voa(X)$, o que é o mesmo que dizer, pela negação, explícita, do próprio predicado:

$$\neg voa(X) \leftarrow avestruz(X)$$

O significado desta última expressão é o de que é ***falso*** que *X* voe, se for uma avestruz. Para já, ganha-se a relação entre as afirmações, que faltava quando representadas num programa em lógica tradicional.

Em [McC80], *John McCarthy* apresenta um exemplo bastante feliz de concretização deste tipo de sistemas. Considere-se a situação em que se pretende permitir que um autocarro escolar atravesse a linha do caminho-de-ferro, na condição de que não se aproxime nenhum comboio. Esta afirmação pode ser representada por:

$$atravessar \leftarrow não\ comboio$$

se a inexistência do átomo $comboio$ na base de conhecimento for interpretada como a ausência do comboio em aproximação. Mas esta convenção de notação é inaceitável se a informação acerca da presença ou ausência do comboio não estiver disponível. Se, por exemplo, o átomo $comboio$ não existir devido ao facto de o condutor do autocarro ter a visão bloqueada, continua a não se pretender que o autocarro atravesse a linha.

No entanto, esta diferença entre a intuição que se tem do caso e o significado da regra representada pode desaparecer, considerando a utilização da negação clássica para descrever a situação:

$$atravessar \leftarrow \neg comboio$$

O significado desta expressão é, agora, de que a conclusão $atravessar$ só é possível se for ***falsa*** a existência do termo $comboio$, ou seja, se for falsa a existência do comboio em aproximação. Desta forma, a conclusão $atravessar$ não fará parte das soluções para a questão a menos que o facto $\neg comboio$ seja incluído na base de conhecimento.

A diferença entre estes dois casos está em que, na primeira situação, o

autocarro atravessará a linha se ***não existir uma prova*** de que o comboio esteja a aproximar-se, e, na segunda, o comboio atravessará se ***existir uma prova*** de que o comboio ***não está*** a aproximar-se.

A distinção entre o significado de $não\ P$ e $\neg P$ é essencial quando não se quer (ou não se pretende) assumir que a informação existente sobre *P* é completa, isto é, quando não se aplica o PMF a *P*, não podendo assumir que a ausência de informação denote, inequivocamente, a sua falsidade.

Para além do mais, a introdução da negação explícita, juntamente com a negação por falha, permite uma maior expressividade da linguagem. Se o problema for o de o condutor do autocarro ***não ter a certeza absoluta*** de que o comboio ***não está a aproximar-se***, então ***não deverá atravessar*** a linha. Esta afirmação pode ser descrita de uma forma muito natural, nomeadamente:

$$\neg atravessar \leftarrow não\ \neg comboio$$

Assim, declarações do tipo "***não*** *é possível mostrar que* ***não*** *possa acontecer P*", podem ser descritas por formulações da forma $não\ \neg P$. Exemplos de situações como esta são comuns em problemas do dia-a-dia, que envolvem o raciocínio por senso comum.

Em sistemas construídos com base nesta forma de extensão à PL, passamos a ter uma terceira hipótese de responder às questões: para além de serem verdadeiras as conclusões obtidas a partir da informação positiva e falsas as obtidas em função da informação negativa, torna-se possível concluir que a resposta é ***desconhecida***, se nenhuma das conclusões anteriores for possível.

Formalmente, podemos definir um programa em lógica estendido como um conjunto de regras da forma:

$$q \leftarrow p_1 \wedge \ldots \wedge p_m \wedge não\ p_{m+1} \wedge \ldots \wedge não\ p_{m+n}$$

em que tanto o q como os p_i são literais, isto é, fórmulas da forma a ou $\neg a$, sendo

a um átomo, e $m, n \geq 0$.

A semântica de um programa em lógica estendido atribui, ao programa, um ***conjunto de respostas***, $\mathcal{R}$, como sendo o conjunto das fórmulas que correspondem às conclusões que podem ser obtidas por um sistema de inferência capaz de implementar o esquema de raciocínio adequado. Diremos que $\neg q$ é ***verdadeiro***, para um determinado conjunto de respostas, $\mathcal{R}$, se $\neg q \in \mathcal{R}$, enquanto que *não* q é verdadeiro, em $\mathcal{R}$, se $q \notin \mathcal{R}$.

A interpretação de uma questão q colocada a um sistema capaz de raciocinar em termos da Programação em Lógica Estendida (PLE), Π, pode ser definida, informalmente, da seguinte forma:

- ***verdadeira*** se q é ***verdadeiro*** em todos os conjuntos de respostas do programa Π;
- ***falsa*** se $\neg q$ é ***verdadeiro*** em todos os conjuntos de respostas de Π;
- ***desconhecida*** em qualquer outro caso.

ou seja, em termos práticos, considerando uma questão abstracta *q(X)*, em que X pode tomar a forma $X_1, X_2, \ldots, X_n$, definem-se as seguintes três respostas possíveis para essa questão:

- ***verdadeiro*** se $\exists_X : q(X)$
- ***falso*** se $\exists_X : \neg q(X)$
- ***desconhecido*** se $\neg \exists_X : q(X) \vee \neg q(X)$

em que '*:*' é a notação para "*tal que*".

Outra importante motivação para a introdução da negação explícita em programas em lógica está relacionada com o paralelismo existente entre informação positiva e negativa. Podem ocorrer situações em que seja mais fácil ou

mais natural expressar directamente a informação negativa e, posteriormente, tirar conclusões em função dessa representação.

Considere-se o exemplo de um programa em lógica que implementa uma descrição de um grafo dirigido, baseado na extensão do predicado `arco(X,Y)`, que exprime o pressuposto de que no grafo existe um arco do nodo `X` para o nodo `Y`, ou seja, o predicado `arco` é constituído por dois argumentos, sendo o primeiro a identificação do nodo origem e o segundo a identificação do nodo destino, denotando uma ligação orientada nesse sentido.

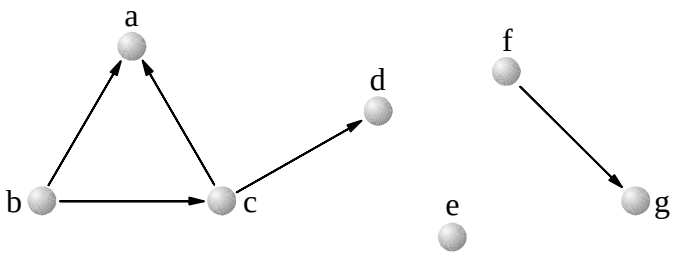


Figura 1: Grafo dirigido com 7 nodos

O grafo da Figura 1 pode ser descrito por um programa em lógica, tal como se mostra no Programa 1.

```
arco(b,a)
arco(b,c)
arco(c,a)
arco(c,d)
arco(f,g)
```

Programa 1: Extensão do predicado `arco`

Supor que, para a construção desse programa, se pretende determinar os

nodos terminais do grafo, ou seja, os nodos a partir dos quais não existe saída. No exemplo descrito pela Figura 1, estão nessa situação os nodos ‘a’, ‘d’, ‘e’ e ‘g’.

Esta é uma situação para a qual é bastante mais simples pensar em termos daquilo que define um nodo que não é terminal, isto é, em vez de se descrever o que é um nodo terminal, pode-se, mais simplesmente, definir o que ***não é*** um nodo terminal: o nodo *X* não é terminal se for o ponto de partida de um arco:

$$\neg terminal(X) \leftarrow arco(X,Y)$$

X é um nodo terminal se não existir uma prova de que seja não terminal:

$$terminal(X) \leftarrow não\ \neg terminal(X)$$

Finalmente, outra importante motivação para a utilização da PL, estendida com a negação explícita, é a generalização da relação entre a PL e os formalismos do raciocínio não-monótono.

A adequação da PLE tem sido posta à prova em várias formas de raciocínio, nomeadamente sobre informação incompleta e raciocínio por defeito, bem como em sistemas de representação de conhecimento, tais como regras de produção, regras por defeito, excepções e representação de informação negativa, quer através da negação explícita, quer da negação por falha.

3 O PMF na Programação em Lógica Estendida

A diferença substancial entre um programa em lógica e um programa em lógica estendido, é que, no primeiro, a única forma de negação utilizada é a negação por falha na prova e a aplicação do PMF a todos os predicados do programa, enquanto que um programa em lógica estendido, para além da negação por falha, faz uso, ainda, da negação explícita para formalizar a descrição de informação, situação que é “*simulada*” pela negação por falha num programa em lógica.

À partida, num programa em lógica estendido, não é aplicado o PMF a nenhum dos predicados.

Sintacticamente, um programa em lógica é, em certa medida, um caso especial de um programa em lógica estendido. Contudo, semanticamente, existem diferenças no significado a atribuir a um conjunto de regras num programa em lógica e aquele a dar às que se encontram no contexto de um programa em lógica estendido.

Atente-se no Programa 2 que apresenta uma extensão do predicado `par`, que permite calcular os números pares.

```
par(0)
par(s(s(X))) ← par(X)
```

Programa 2: Extensão do predicado `par`

Em termos da PLE, o conjunto de soluções que se obtém a partir da extensão do predicado do Programa 2 é dado por:

$$\{\texttt{par(0)}, \texttt{par(s(s(0)))}, \texttt{par(s(s(s(s(0)))))}, ...\}$$

Isto significa que, uma vez que não contém a declaração `par(s(0))` nem $\neg$`par(s(0))`, a resposta à questão `par(s(0))` será ***desconhecido***, o que não é bem o que se pretende expressar neste caso.

Nesta situação, o que pretendemos expressar pela ausência de uma solução em particular não é a de que esta seja desconhecida, mas, antes, de que essa solução, de facto, é falsa (porque não existe), ou seja, pretenderíamos ver aqui aplicado o PMF, para que a ausência de informação fosse considerada como a negação da sua existência.

Este significado pode ser contemplado no predicado descrito pelo programa

em lógica estendido anterior se lhe adicionarmos uma regra que formalize o conceito subjacente ao PMF, expressando que é falso tudo aquilo que não for possível provar como sendo verdadeiro. Neste caso, é falso que um número seja par se não existir uma prova de que o seja:

```
¬par(X) ← não par(X)
```

Incluindo esta regra no Programa 2, o predicado `par(X)` toma a forma apresentada no Programa 3.

```
par(0)
par(s(s(X))) ← par(X)
¬par(X) ← não par(X)
```

Programa 3: Formalização do PMF para o predicado `par`

Desta maneira, o conjunto de soluções representado pelo Programa 3 é

$$\{\texttt{par(0)}, \neg\texttt{par(s(0))}, \texttt{par(s(s(0)))}, \ldots\}$$

Como já havia sido referido, a ausência de um determinado átomo *A*, num programa em lógica, significa que *A* é falso e que a resposta à questão deverá ser ***falsa***, enquanto que a falta desse mesmo átomo *A* no conjunto de soluções de um programa em lógica estendido, formado pelo mesmo conjunto de regras, indica que a resposta a essa questão deva ser ***desconhecido***.

Este exemplo sugere que um programa em lógica é semanticamente equivalente a um programa em lógica estendido ao qual se adicionam regras com a formalização do PMF para cada um dos predicados existentes.

Genericamente, pode-se definir o PMF para um predicado p, na linguagem de um programa em lógica estendido, pela regra:

$$\neg p(X) \leftarrow não\ p(X) \qquad (1)$$

em que X pode tomar a forma X_1, X_2, …, X_n, sendo n o número de argumentos do predicado p.

Um programa em lógica estendido, ao qual se adicionam regras como (1) para cada um dos predicados existentes no programa, transforma-se num programa em lógica estendido semanticamente equivalente a um programa em lógica.

Refira-se que um sistema capaz de raciocinar em termos de informação incompleta tal como foi apresentado nesta secção, não é, necessariamente, um sistema não-monótono, no âmbito do que foi discutido em [Ana96].

4 Valores Nulos

Nas secções anteriores foi abordada a problemática do raciocínio em face de falta de informação ou onde se supõe a representação parcial de informação.

Nesta secção pretende-se fazer um estudo do tipo de valores mais comuns que podem surgir numa situação de informação incompleta. Esta identificação de valores, designados por ***valores nulos***, surge como uma estratégia para a enumeração de casos, para os quais se pretende fazer a distinção entre situações em que as respostas a questões deverão ser concretizadas como ***conhecidas*** (verdadeiras ou falsas) ou ***desconhecidas*** [TrGe93].

A representação de valores nulos será abordada no âmbito da PLE que tem vindo a ser apresentada neste capítulo. Serão três os valores nulos aqui tratados: o primeiro permitirá representar valores desconhecidos e não necessariamente de um conjunto determinado de valores; o segundo representará valores desconhecidos, mas de um conjunto ~~finito e~~ determinado de valores; por fim, o terceiro género de valores nulos será de um tipo ligeiramente diferente e representará valores não permitidos, considerados, nomeadamente, na assimilação de informação na base de conhecimento.

Nas subsecções seguintes faz-se uma apresentação pormenorizada destes tipos de valores nulos.

4.1 Do Tipo Desconhecido

Considere-se o caso onde se pretende identificar a forma de canto de algumas aves. A Tabela 1 representa a totalidade da informação conhecida, ou seja, transcreve ***completamente*** o conhecimento que se tem, neste momento, sobre o canto das aves.

Tabela 1: Forma de canto das aves

AVE	*SOM*
periquito	*chilro*
canário	*assobio*

Esta descrição informal do conhecimento pode ser representada através PLE, pela introdução do predicado `canto` com dois argumentos, o primeiro representando a *AVE* e o segundo identificando o *SOM* que esta emite. A informação contida na Tabela 1 pode ser representada pelo programa em lógica dado pelo Programa 4.

A terceira cláusula do Programa 4 representa a formalização do PMF para o predicado `canto`, o que se justifica, neste caso, por se ter assumido que a Tabela 1 representa completamente a informação do problema, e permite concluir, correctamente, que, por exemplo, `¬canto(canário,chilro)`, ou seja, que é falso que o canto do canário seja o chilro.

```
canto(periquito,chilro)
canto(canário,assobio)
¬canto(Ave,Som) ←
     não canto(Ave,Som)
```

Programa 4: Representação formal da informação da Tabela 1

No entanto, a situação modifica-se se a Tabela 1 for substituída pela Tabela 2.

Tabela 2: Expansão à informação da Tabela 1

AVE	*SOM*
periquito	*chilro*
canário	*assobio*
ave rara	*fado*

Nesta tabela, a entidade *ave rara*[1] representa um valor específico do tipo *AVE*. Este é um ***valor nulo*** do tipo ***desconhecido e não necessariamente de um conjunto determinado de valores***, que representa uma ave ***desconhecida***, possivelmente mesmo, diferente de *periquito* ou *canário*, que são as duas únicas aves concretizadas na descrição do problema.

Neste contexto, a resposta à questão `canto(X,assobio)` – qual é a ave cujo canto é o assobio – deverá dar como solução a associação à variável `X` do valor atómico `canário`, enquanto que a resposta à questão `canto(X,fado)` –

[1] O sublinhado da expressão é, apenas, um pormenor gráfico para melhor identificar o termo como um caso especial, não sendo, contudo, essencial para a ilustração da situação.

qual é a ave cujo canto é o fado – deverá ser ***desconhecida***, no sentido em que a questão é verdadeira, porque tem solução, mas a concretização dessa solução é que é desconhecida, já que não se sabe de nenhuma ave em concreto nessas condições; apenas se sabe que existe uma ave rara que canta o fado.

Em continuação, a resposta à questão `canto(canário,chilro)` continuará a ser ***falsa***, enquanto que a questão `canto(canário,fado)` não terá uma resposta conclusiva por ser ***desconhecida*** a ave que canta o fado.

Após esta análise superficial ao problema proposto, e considerando a informação da Tabela 2, parece óbvio que se represente essa informação acrescentando ao Programa 4 a cláusula:

```
canto(ave rara,fado)
```

Contudo, esta alteração ainda não permite obter as conclusões desejadas, já que a questão `canto(canário,fado)` é ***falsa*** por não existir uma prova de que seja verdadeira (através da cláusula de formalização do PMF) quando, em função do exposto, se pretenderia que fosse ***desconhecida***.

Aquilo que está a acontecer é que a conclusão obtida a partir da cláusula que formaliza o PMF para o predicado `canto` não está a permitir a flexibilização das respostas por não admitir excepções. E a questão é mesmo esta: o valor nulo que identifica a ave rara que canta o fado deverá ser considerado como uma excepção à regra de formalização do PMF.

Para representar correctamente esta informação torna-se necessária uma formalização adequada do PMF numa linguagem de PL que permita tratar valores nulos.

O que está em causa neste momento é a mudança de estado da base de conhecimento entre o instante que definiu a informação contida na Tabela 1 e o instante que deu origem à informação da Tabela 2, ou seja, o problema está relacionado com a flexibilidade do sistema aquando da evolução do conhecimento.

Estas questões foram abordadas em [Ana96], onde se faz referência à representação de excepções como situações anómalas em relação à extensão de um determinado predicado.

Consideremos, então, que a identificação de valores nulos do tipo desconhecido será feita através da introdução de uma situação de excepção correspondendo a uma condição anómala, na cláusula de formalização do PMF. Assim, a terceira cláusula do Programa 4 deverá ser substituída por:

```
¬canto(Ave,Som) ←
     não canto(Ave,Som) ∧
     não excepção_canto(Ave,Som)
```

e deverá ser identificada uma excepção relativamente ao canto das aves na forma:

```
excepção_canto(Ave,Som) ←
     canto(ave rara,Som)
```

Com esta formulação, pretendemos definir que é ***falso*** que o `canto` da `Ave` seja um dado `Som` se acontecer não existir uma prova de que assim seja e, ainda, não existir uma prova de que haja uma situação anómala definida entre essa `Ave` e esse `Som` para o `canto` daquela.

O programa resultante é o que se mostra no Programa 5, que expressa correctamente a informação representada na Tabela 2.

Generalizando, vimos que valores nulos do tipo desconhecido se representam, no contexto de um programa em lógica estendido, como situações anómalas a identificar na definição dos casos que são considerados excepções à concretização da informação negativa.

Para um predicado p, representa-se um valor nulo v_i do tipo desconhecido, como uma excepção a $\neg p$:

$$\neg p(X) \leftarrow não\ excepção_p(X)$$

```
canto(periquito,chilro)
canto(canário,assobio)
canto(ave rara,fado)
¬canto(Ave,Som) ←
     não canto(Ave,Som) ∧
     não excepção_canto(Ave,Som)

excepção_canto(Ave,Som) ←
     canto(ave rara,Som)
```

Programa 5: Representação da informação da Tabela 2

concretizando, de seguida, as excepções que se pretendem definir como valores nulos:

$$excepção_p(X) \leftarrow p(v_1)$$

$$excepção_p(X) \leftarrow p(v_2)$$

...

Nestas expressões, considerando X como uma sequência de variáveis $X_1, X_2, \ldots, X_n$, em que n é o número de argumentos de p, então v_i representará a mesma sequência, mas na qual se substituirá a variável X_j pelo valor nulo específico v, por exemplo, $X_1, \ldots, v, \ldots, X_n$.

No caso de aplicação que tem sido utilizado, tal foi efectuado incluindo a excepção no corpo da cláusula de formalização do PMF que já fazia parte do programa em lógica, correspondendo à forma genérica:

$$\neg p(X) \leftarrow não\ p(X) \wedge não\ excepção_p(X)$$

nas mesmas condições que foram enunciadas para as expressões anteriores.

4.2 Do Tipo Desconhecido, de um Conjunto Dado de Valores

A diferença entre o valor nulo do tipo desconhecido e não necessariamente de um conjunto determinado de valores, visto na subsecção anterior, e o valor nulo que se pretende apresentar agora, ***desconhecido mas de um conjunto ~~finito e~~ determinado de valores***, está precisamente neste facto: este último valor nulo representará um (ou mais) valores de um conjunto ~~finito~~ de valores bem determinados; só não é conhecido, especificamente, qual dos valores concretizará a questão.

Considere-se uma extensão à Tabela 2 através da asserção de que o *SOM* do canto da *AVE pinguim* é uma de duas hipóteses: *ópera* ou música *clássica*. Representemos esta informação pela Tabela 3.

Tabela 3: Extensão com informação sobre os pinguins

AVE	*SOM*
periquito	*chilro*
canário	*assobio*
ave rara	*fado*
pinguim	*{ópera, clássica}*

A expressão *{ópera, clássica}* representa o conjunto finito de possíveis valores para concretizar o tipo de informação sobre o *SOM* do *pinguim*.

Numa primeira aproximação, pode-se representar esta nova porção de informação através de uma disjunção de termos:

$$\texttt{canto(pinguim,ópera)} \vee \texttt{canto(pinguim,clássica)} \qquad (2)$$

e incluir esta declaração no Programa 5. Semanticamente, a base de conhecimento resultante representa correctamente a informação da Tabela 3. Contudo, a introdução da disjunção de termos não é, normalmente, implementada em sistemas baseados na PLE, pelo menos directamente. Quando isso é feito, paga-se um custo computacional elevado pelo mecanismo de inferência sobre bases de dados disjuntivas.

Sendo assim, torna-se necessário reformular esta aproximação por forma a tornear esta desvantagem, o que será conseguido através de uma abordagem semelhante à realizada para o caso de valores nulos apresentado anteriormente.

A informação representada pela expressão (2) não pode ser utilizada considerando isoladamente cada um dos termos da disjunção uma vez que não se sabe, concretamente, qual das partes é verdadeira. No entanto, pode ser considerada como uma excepção em relação àquilo que é negado ser verdadeiro.

Relativamente à identificação das excepções à cláusula que formaliza o PMF para o predicado `canto`, esta situação envolve um caso muito particular de anormalidades. Ser possível o pinguim cantar ópera é uma excepção à negação de que o faça, tal como o é o facto de poder cantar música clássica. Deste modo serão construídas duas excepções, correspondentes a outras tantas possibilidades de resposta:

$\texttt{excepção}_{\texttt{canto}}\texttt{(pinguim,ópera)}$
$\texttt{excepção}_{\texttt{canto}}\texttt{(pinguim,clássica)}$

Este tipo de excepções não se verificam se o pinguim cantar ou não cantar o que quer que seja. São, simplesmente, excepções às conclusões de que o não faça. Acrescentando estas asserções ao Programa 5, obtemos como resultado o Programa 6, capaz de lidar com o tipo de informação disjuntiva, tal como era o caso da expressão (2).

```
canto(periquito,chilro)
canto(canário,assobio)
canto(ave rara,fado)
¬canto(Ave,Som) ←
     não canto(Ave,Som) ∧
     não excepção_canto(Ave,Som)

excepção_canto(Ave,Som) ←
     canto(ave rara,Som)
excepção_canto(pinguim,ópera)
excepção_canto(pinguim,clássica)
```

Programa 6: Representação de informação disjuntiva

Em virtude da interpretação de um programa em lógica estendido, apresentada na secção 2 Extensão à Programação em Lógica, e uma vez que não existe informação positiva que o declare, não se pode concluir que o pinguim cante ópera (ou música clássica), mas, como existem excepções que impedem que a conclusão seja falsa, a resposta a qualquer uma das questões `canto(pinguim,ópera)` ou `canto(pinguim,clássica)` deverá ser desconhecida. Sem embargo, a resposta à questão `canto(pinguim,assobio)`, por exemplo, continuará a ser falsa, uma vez que o pinguim não canta qualquer coisa, só canta ópera ou música clássica, de resto, tal como está declarado no Programa 6.

Concluindo, podemos dizer que valores nulos do tipo analisado nesta subsecção, representam-se como um conjunto de excepções específicas à identificação da informação negativa representada na base de conhecimento.

Para um predicado p, define-se a representação de um conjunto de valores nulos v_i do tipo desconhecido, de um conjunto determinado de valores, como uma

sequência de excepções a $\neg p$, definido por:

$$\neg p(X) \leftarrow não\ excepção_p(X)$$

ou, como no caso de aplicação presente, com a forma:

$$\neg p(X) \leftarrow não\ p(X) \wedge não\ excepção_p(X)$$

concretizadas como excepções específicas, cada uma para cada um dos valores nulos que são possíveis soluções de representação do valor que se desconhece:

$$excepção_p(v_1)$$

$$excepção_p(v_2)$$

...

Os v_i representam os valores nulos do conjunto de valores possíveis existentes, sendo valores atómicos que concretizam explicitamente todas as possibilidades de representação dos valores nulos.

4.3 Do Tipo Não Permitido

O terceiro género de valores nulos que abordaremos de seguida, são ligeiramente diferentes de ambos os que foram considerados anteriormente, uma vez que, além de identificarem, também, valores desconhecidos, caracterizam, ainda, um tipo de dados que não se pretende admitir que surjam na base de conhecimento.

Para se apresentar este valor nulo, e, na sequência do conhecimento que tem vindo a ser descrito, considere-se a informação representada na Tabela 4.

O valor representado por ω, para além de identificar um valor ***desconhecido***, pretende significar que o valor que representa ***não é permitido*** especificar ou conhecer, no caso de identificar aves que cantem o hino, e qualquer tentativa posterior de concretizar esse valor deverá ser rejeitada como sendo provocadora de inconsistência na informação constante da base de conhecimento. Isto significa

Tabela 4: Introdução de valores do tipo não permitido

AVE	*SOM*
periquito	*chilro*
canário	*assobio*
ave rara	*fado*
pinguim	*{ópera, clássica}*
ω	*hino*

que a última expansão à base de conhecimento, de que resultou a Tabela 4, introduziu a informação de que existe uma ave, cuja identificação específica não se permite conhecer, que canta o hino.

Se começarmos por considerar este valor nulo como sendo do mesmo tipo do que foi apresentado na secção 4.1, desconhecido e não necessariamente de um conjunto determinado de valores, podemos representá-lo na base de conhecimento, na forma:

```
canto(ω,hino)
```

e considerá-lo como uma excepção à verificação da informação negativa do predicado `canto`, isto é:

```
excepção_canto(Ave,Som) ←
    canto(ω,Som)
```

Desta forma, por exemplo, a questão `canto(periquito,chilro)` continua a ser verdadeira (outra coisa não seria de esperar), a questão `canto(canário,chilro)` continuará a ter resposta falsa, enquanto que a resposta à questão `canto(canário,hino)` será desconhecida, uma vez que não

se conhece (nem se permite conhecer) o valor concreto da ave que canta o hino.

O programa resultante, que corresponde a esta última expansão da informação, existente na Tabela 4, está representado no Programa 7.

```
canto(periquito,chilro)
canto(canário,assobio)
canto(ave rara,fado)
canto(ω,hino)
¬canto(Ave,Som) ←
     não canto(Ave,Som) ∧
     não excepção_canto(Ave,Som)

excepção_canto(Ave,Som) ←
     canto(ave rara,Som)
excepção_canto(pinguim,ópera)
excepção_canto(pinguim,clássica)
excepção_canto(Ave,Som) ←
     canto(ω,Som)
```

Programa 7: Representação de valores nulos do tipo não permitido

O Programa 7 representa correctamente o conhecimento estático a que corresponde a informação da Tabela 4. No entanto, esta representação não é suficiente para capturar o significado dinâmico que se pretende associar aos valores nulos do tipo não permitido, cujas tentativas posteriores de alteração deverão ser impedidas.

Considere-se que é "*forçada*" a inserção na base de conhecimento do termo:

```
canto(papagaio,hino)                    (3)
```

É claro que não é provocado nenhum problema de inconsistência com a informação que já existia na base de conhecimento. Somente, a noção intuitiva que

tínhamos da representação do valor nulo não permitido não está a ser observada, porque a partir deste momento existiria a identificação de uma ave – o `papagaio` – que cantaria o hino.

Se é a nossa intuição que não está a ser observada pela informação da base de conhecimento, é porque ainda não concretizamos nenhum formalismo capaz de a representar.

Mais uma vez o problema ocorre durante o processo de evolução da informação na base de conhecimento. No caso em discussão, o que se pretende é que não exista qualquer evolução se ela se vier a efectuar através da inclusão de informação que viole a condição imposta pelo valor nulo não permitido.

A questão está em identificar a condição presente num valor não permitido. Se considerarmos uma regra de teste à consistência da informação da base de conhecimento, essa regra terá de exprimir que uma *AVE*, cujo *CANTO* é um determinado *SOM*, no caso presente o *hino*, só deverá existir explicitamente se essa *AVE* não for um valor nulo do tipo não permitido.

Comecemos por considerar a representação da informação que identifique o valor nulo do tipo não permitido, introduzindo o predicado `nulo`, com um argumento apenas, que representará, especificamente, o valor nulo. O facto de ω ser um valor com estas características seria representado por:

```
nulo(ω)
```

A existência de uma fórmula de verificação da consistência da informação da base de conhecimento com o significado pretendido, poderá ser descrita por:

```
canto(Ave,hino) ∧ não nulo(Ave)                    (4)
```

Esta fórmula significa que se encontra uma inconsistência, se existir uma ave que cante o hino e que não esteja identificada como sendo um valor nulo. O objectivo é que, sempre que aplicada, esta fórmula nunca seja capaz de gerar

qualquer solução, o que significaria que nunca seria encontrada nenhuma inconsistência. Formalmente, em PL, a fórmula (4) deve ser representada por uma regra cuja conclusão seja sempre falsa, isto é, correspondendo a uma cláusula com a cabeça vazia, da forma:

$$\leftarrow \text{canto(Ave,hino)} \wedge \text{não nulo(Ave)} \qquad (5)$$

O programa em lógica que implementa correctamente a informação representada na Tabela 4, deverá representar, então, o valor nulo ω como tal, bem como incluir a regra (5), que representa a restrição que captura o significado dinâmico do valor nulo, para além da formulação que já havia sido proposta no Programa 7. O programa em lógica resultante é o que se ilustra no Programa 8.

```
canto(periquito,chilro)
canto(canário,assobio)
canto(ave rara,fado)
canto(ω,hino)
¬canto(Ave,Som) ←
     não canto(Ave,Som) ∧
     não excepção_canto(Ave,Som)

excepção_canto(Ave,Som) ←
     canto(ave rara,Som)
excepção_canto(pinguim,ópera)
excepção_canto(pinguim,clássica)
excepção_canto(Ave,Som) ←
     canto(ω,Som)

nulo(ω)

← canto(Ave,hino) ∧ não nulo(Ave)
```

Programa 8: Representação completa de valores nulos do tipo não permitido

Neste momento, se tentarmos, novamente, "*forçar*" a inclusão da expressão (3) na base de conhecimento, será claramente rejeitada uma vez que viola a restrição representada pela expressão (5) e que está incluída no programa em lógica.

Imaginemos que, uma vez que não existe qualquer restrição imposta no Programa 8 à introdução de valores atómicos em termos do predicado `nulo`, inserimos na base de conhecimento, a asserção:

```
nulo(papagaio)                                                    (6)
```

que, isoladamente, parece não ter qualquer significado particular. Após isto, se tentarmos a inserção da informação representada pela expressão (3), esta é bem sucedida, ou seja, acabamos de inserir a informação de que o papagaio canta o hino. Parece que o caldo se voltou a entornar!...

Mas será que é mesmo assim? Não! Ao inserir a expressão (3) após a inclusão de (6), o que se introduz na base de conhecimento é um valor nulo e não um facto concreto, ou seja, o que se incluiu na base de conhecimento foi mais um valor nulo do tipo não permitido, tal como já acontecia com o valor ω, mas, desta vez, com a identificação `papagaio`. Assim, o átomo `papagaio` representa um valor nulo não permitido e não a identificação de uma ave. O resultado é que ficamos com informação sobre dois valores nulos, do tipo não permitido, que, precisamente por representarem valores desconhecidos, acabam por ter exactamente o mesmo significado.

O único problema que pode ocorrer com esta situação é o de se estar a gerar informação redundante na base de conhecimento, mas isso é uma questão a ser resolvida por outros mecanismos que não os de representação de valores nulos que estão a ser considerados.

Podemos considerar que a componente estática do significado deste tipo de valores nulos é representada exactamente da mesma forma que os valores nulos do

tipo desconhecido apresentados na secção 4.1, recorrendo à identificação de situações anómalas definidas como excepções. O significado dinâmico de não permitir qualquer actualização a este tipo de valores é traduzido pela sua identificação explicita de valor nulo, a confirmar pela aplicação de uma restrição adequadamente construída para o efeito.

Para um predicado p, representa-se um valor nulo v_i do tipo não permitido, como uma excepção a $\neg p$, definido por:

$$\neg p(X) \leftarrow não\ excepção_p(X)$$

ou por:

$$\neg p(X) \leftarrow não\ p(X) \wedge não\ excepção_p(X)$$

para o caso de estudo que está a ser considerado, para a qual se concretizam os valores nulos na forma:

$$excepção_p(X) \leftarrow p(v_1)$$

$$excepção_p(X) \leftarrow p(v_2)$$

$$\ldots$$

sendo X uma sequência de variáveis $X_1, X_2, \ldots, X_n$, e em que cada v_i representará a mesma sequência, na qual se substitui a ocorrência particular de uma variável X_j pelo valor nulo específico v, por exemplo, $X_1, \ldots, v, \ldots, X_n$.

Em continuação, cada um dos valores nulos v_i deve ainda ser representado explicitamente como um valor nulo, na forma:

$$nulo(v_1)$$

$$nulo(v_2)$$

$$\ldots$$

e incluída um restrição que impeça a posterior inclusão de valores atómicos que violem a condição imposta pelo valor nulo não permitido:

$$\leftarrow p(X) \wedge não\ nulo(X)$$

De notar que, para cada predicado p, pode existir mais do que um valor nulo do tipo não permitido com significados diferentes. Por exemplo, pode existir um ω_1 que cante o hino, um ω_2 que cante mal, um ω_3 que cante outra coisa qualquer, e assim por diante.

5 Interpretação de Valores Nulos

Neste ponto torna-se útil apresentar um sistema que, por muito simples que seja, permita implementar mecanismos de inferência sobre estes três tipos de valores nulos, que concretizam situaçõ es de informação incompleta.

Tal sistema alicerçar-se-á no paradigma da PL, suportando as três respostas possíveis neste contexto: respostas dos tipos ***verdadeira***, ***falsa*** e ***desconhecida***.

A implementação prática deste sistema não será mais do que um interpretador para questões colocadas em termos de um programa em lógica estendido, interpretação essa que será efectuada nos termos apresentados na secção 2 Extensão à Programação em Lógica. Esse interpretador será traduzido pela extensão de um meta-predicado que interpretará a questão colocada ao sistema em termos da sua especificidade.

Em primeiro lugar há que definir a estrutura do predicado que se designará por `demo` (de `demo`nstração). Deste predicado, e como argumentos, ter-se-ão a questão a demonstrar e o respectivo resultado, que estará entre um de três valores possíveis: a veracidade, falsidade ou o desconhecimento de resposta para a questão. O predicado pode, então, ser definido nos termos:

`demo: Questão, Resposta` $\rightarrow \{\mathbb{V}, \mathbb{F}\}$

isto é, o predicado `demo` é constituído por dois argumentos, sendo o primeiro

respeitante à `Questão` a colocar ao sistema e o segundo a `Resposta` para a questão colocada. Uma vez que a extensão deste predicado faz parte de um programa em lógica, tem como contradomínio um dos dois valores de verdade: *verdadeiro*, $\mathbb{V}$, ou *falso*, $\mathbb{F}$.

Tendo-se em linha de conta o que já foi mencionado sobre a interpretação de questões num programa em lógica estendido, obtém-se como extensão do predicado `demo` o programa em lógica apresentado no Programa 9.

```
demo(Q,verdadeira) ←
     Q
demo(Q,falsa) ←
     ¬Q
demo(Q,desconhecida) ←
     não ¬Q ∧
     não Q
```

Programa 9: Interpretador de questões

A extensão do predicado `demo` é dada pelas cláusulas referidas no Programa 9. A primeira indica que a resposta a uma questão `Q` é do tipo `verdadeira` se for possível desenvolver uma prova de `Q` a partir da informação positiva existente na base de conhecimento. A segunda cláusula define que a resposta é do tipo `falsa` se for possível provar `¬Q`. A resposta a `Q` será do tipo `desconhecida` se `não` existir quer uma prova de `¬Q` quer uma prova de `Q`.

Nestas condições, pode-se utilizar a extensão do predicado `demo` dada pelo Programa 9 para dar resposta a algumas das questões que se podem colocar a um sistema baseado em PLE, como é o caso dos exemplos que foram sendo apresentados ao longo da secção 4.

Considere-se o Programa 4, que descreve a informação dada pela Tabela 1.

Uma vez que para o único predicado do Programa 4, `canto`, se aplicou o conceito do PMF, qualquer questão que não seja possível provar em termos das duas primeiras cláusulas (o `periquito` cujo `canto` é o `chilro` e o `canário` cujo `canto` é o `assobio`) deverá ser dada como sendo do tipo falsa.

Suponha-se que se coloca a questão:

```
demo(canto(canário,chilro),Resposta)        (7)
```

É fácil chegar à conclusão de que a `Resposta` é do tipo `falsa`. Primeiro, porque a partir da primeira cláusula do predicado `demo` do Programa 9, não há qualquer possibilidade de fazer a prova através da informação positiva dada no Programa 4. Segundo, porque a partir da segunda cláusula do predicado `demo` é possível provar que é do tipo `falsa` a questão `canto(canário,chilro)` (por não ser possível provar ser do tipo `verdadeira`, em virtude do PMF para o predicado `canto` do Programa 4).

Já no que diz respeito à informação da Tabela 2 e ao Programa 5, onde se trata de valores nulos do tipo desconhecido (subsecção 4.1), a colocação da questão (7) dará os mesmos resultados pelas razões já explicitadas, enquanto que para a questão:

```
demo(canto(canário,fado),Resposta)          (8)
```

através de um procedimento análogo ao do caso anterior, se pode concluir que a `Resposta` é do tipo `desconhecida`. Não é possível desenvolver uma prova para a questão `canto(canário,fado)`, fazendo uso da primeira cláusula do Programa 9, por inexistência de informação que seja válida. Através da segunda cláusula do mesmo programa é cancelada a prova a partir do momento em que se identifica a excepção que corresponde ao facto de o `fado` ser cantado por uma `ave rara` que não se conhece. A terceira cláusula resulta na obtenção de uma `Resposta` do tipo `desconhecida`, por não ser possível desenvolver uma prova em que a questão seja dada como sendo do tipo `verdadeira` nem do tipo `falsa`.

Uma situação idêntica ocorre com a questão:

```
demo(canto(X,assobio),Resposta)
```

cuja `Resposta` é do tipo `verdadeira`, correspondendo à instanciação da variável `X` ao valor atómico `canário`, e do tipo `falsa` nas restantes situações, uma vez que não existe qualquer excepção a assinalar neste caso.

Avançando para o caso de valores nulos do tipo desconhecido, mas de um conjunto determinado de valores, apresentados na subsecção 4.2 e concretizados pelo Programa 6, pode-se observar que tanto a questão:

```
demo(canto(pinguim,ópera),Resposta)
```

como a questão:

```
demo(canto(pinguim,clássica),Resposta)
```

terão uma `Resposta` do tipo `desconhecida`: as questões não se enquadram na prova através da primeira cláusula do Programa 9 por não existir informação positiva que a concretize; não se aplica a segunda cláusula do mesmo programa por existirem excepções definidas que o impedem; resta a aplicação da terceira cláusula, que determina que a `Resposta` seja do tipo `desconhecida`.

Já para a questão:

```
demo(canto(pinguim,assobio),Resposta)
```

a `Resposta` será do tipo `falsa`, já que as excepções não tratam este caso, pelo que se torna possível obter uma prova a partir da segunda cláusula do predicado `demo`. Uma situação semelhante se passa com a questão:

```
demo(canto(periquito,ópera),Resposta)
```

A `Resposta` a esta questão é do tipo `falsa` por motivos idênticos aos enunciados para a questão anterior.

No que concerne aos valores nulos do tipo não permitido, abordados na

subsecção 4.3, são, no que respeita à interpretação estática do conhecimento, idênticos aos valores do tipo desconhecido. Considere-se o Programa 8 que representa a informação necessária para tratar esses casos.

A questão:

```
demo(canto(canário,hino),Resposta)
```

tem o mesmo tipo de interpretação que a questão (8), uma vez que o `hino` é cantado por uma ave desconhecida, resultando numa `Resposta` do tipo `desconhecida`.

Apenas em face de problemas de assimilação de conhecimento é que se justifica a utilização deste tipo de valor nulo, já que a não permissão de se conhecer se refere, acima de tudo, a eventuais alterações à base de conhecimento.

A temática da assimilação de conhecimento é tratada com bastante detalhe por *Robert Kowalski* em trabalhos como [Kow90] e [Kow95], sendo, também, abordada por Paulo Novais em [Nov96].

6 Resumo

Neste capítulo é feita uma abordagem a sistemas de representação de conhecimento que contextualizam situações em que não existe informação completa acerca do problema em equação.

Faz-se uma abordagem à PL como ferramenta para a representação de conhecimento, tornando-se necessário considerar a introdução de uma extensão a esta forma de representação, de modo a ultrapassar a problemática do tratamento de informação incompleta.

A extensão à PL faz-se pela introdução, na linguagem de representação, de dois tipos de negação: a negação por falha na prova, denotada pelo termo *não*, e a

negação forte ou clássica, denotada por '¬'. Neste contexto, passa a ser possível distinguir situações que são falsas de situações para as quais não existe prova de que sejam verdadeiras.

É tratado, ainda, o problema do PMF na PLE, abordando possíveis inconvenientes que a formalização do conceito do PMF possa trazer à extensão de predicados de um programa em lógica estendido.

De seguida faz-se uma abordagem a diferentes tipos de valores que podem estar presentes em sistemas onde se verifiquem situações de informação incompleta, designados por valores nulos. São considerados três géneros de valores nulos: do tipo desconhecido, do tipo desconhecido mas de um conjunto determinado de valores e do tipo não permitido.

Para os valores nulos apresentados, desenvolve-se um programa em lógica estendido (Programa 8) que permite o tratamento desses valores nos termos já enunciados.

Por fim, apresenta-se o desenvolvimento de um meta-predicado (Programa 9) para a manipulação de programas como os mencionados ao longo do capítulo, tendo-se concretizado a sua utilização através de questões introduzidas durante a apresentação dos diversos tipos de valores nulos.

7 Referências

[Ana96] Analide, C., 1996
"Representação de Conhecimento e Raciocínio em Estruturas Hierárquicas",
Tese de Dissertação de Mestrado, Departamento de Informática, Universidade do Minho, Braga, Portugal

[Hus94] Hustadt, U.,1994
"Do we need the closed-world assumption in knowledge representation?",
in Working Notes of the KI'94 Workshop, pp. 24-26, Baader, Buchheit, Jeusfeld, Nutt (eds.), Saarbrücken, Alemanha

[Kow90] Kowalski, R. A., 1990
"Problems and Promises of Computational Logic",
in Proceedings of the Symposium on Computational Logic, Lloyd (ed.), pp. 80-95, Springer Verlag Lecture Notes in Computer Science

[Kow95] Kowalski, R. A., 1995
"Using Meta-Logic to Reconcile Reactive with Rational Agents",
in Meta-Logic and Logic Programming, Apt and Turini (eds), pp. 227-242, MIT Press

[McC80] McCarthy, J., 1980
"Circumscription – A Form of Non-Monotonic Reasoning",
in Journal of Artificial Intelligence, 13 (1,2), pp. 27-39

[Nov96] Novais, P., 1996
"Sistema de Diagnóstico Multi-Agente",
Tese de Dissertação de Mestrado, Departamento de Informática, Universidade do Minho, Braga, Portugal

[TrGe93] Traylor, B., Gelfond, M., 1993
"Representing Null Values in Logic Programming",
in Proccedings of the ILPS'93 Workshop on Logic Programming with Incomplete Information, pp. 35-47, Vancouver, Canadá