Tecnologia para Sistemas Inteligentes
Apontamentos para as aulas sobre

Analogia: Representação e Raciocínio Baseado em Casos

Luís Miguel Botelho

Departamento de Ciências e Tecnologias da Informação
Instituto Superior de Ciências do Trabalho e da Empresa

Abril de 2012

Tecnologias para Sistemas Inteligentes

Apontamentos para as aulas

Índice

Analogia: Representação e Raciocínio Baseado em Casos

O Raciocínio Baseado em Casos (*CBR, Case Based Reasoning*) é a forma mais divulgada de raciocínio por analogia, um dos tipos de raciocínio mais usados pelas pessoas. Em vez de recorrer a regras gerais sobre o domínio do problema a resolver, o raciocínio por analogia estabelece uma relação de semelhança entre o problema a resolver e problemas anteriores já resolvidos. O objectivo é que essa relação de analogia possa servir como base para adaptar a solução usada nos problemas passados de modo que possa ser usada para resolver os novos problemas.

As pessoas usam o raciocínio por analogia em diversos tipos de actividade, por exemplo em actividades de gestão, de direito e de diagnóstico.

Um médico que observa um paciente com determinados sintomas pode recordar-se de tratamentos bem sucedidos usados em doentes com sintomas semelhantes. Um engenheiro que observa um equipamento com mau funcionamento pode recordar-se do diagnóstico feito em casos semelhantes que tenha observado no passado.

Um gestor que pretende fazer um orçamento pode recorrer ao orçamento do ano anterior e adaptá-lo para o ano actual. Quem pretende fazer um caderno de encargos para um concurso de aquisição de bens ou de serviços pode procurar adaptar cadernos de encargos usados noutras ocasiões. Os horários das aulas, numa escola, são quase sempre adaptados de horários anteriores.

Em certos sistemas jurídicos, como por exemplo no Americano, as decisões de um tribunal baseiam-se muitas vezes nas decisões tomadas, no passado, em casos semelhantes.

O CBR (Raciocínio Baseado em Casos) foi proposto por Janet Kolodner [Kolodner 1993] como uma ferramenta conceptual e computacional de representação e raciocínio para sistemas de Inteligência Artificial. O CBR foi uma técnica muito aprofundada por outros investigadores de Inteligência Artificial. Por exemplo, Jaime Carbonell e especialmente Manuela Veloso [Veloso, Mulvehill and Cox 1997][Veloso 1997][Veloso and Carbonell 1993] usaram CBR em sistemas inteligentes que planeiam a sua acção.

Enquanto que a lógica e as regras representam conhecimento geral, o raciocínio por analogia baseado em casos, representa casos particulares. Seguidamente, apresentam-se exemplos que procuram clarificar a diferença entre conhecimento geral e conhecimento baseado em casos e a forma como são usados.

Conhecimento geral

IF Heroi(H) AND Vilao(V) AND Pos(H, Pos) AND Pos(V, Pos) THEN Disparar(H, V)

Neste jogo, isto é uma regra aplicável a qualquer herói e a qualquer vilão, sejam eles quais forem, que se encontrem em qualquer posição do mundo do jogo

$\forall x$ Numero(x) $\wedge$ Inteiro(x) $\wedge$ x Mod 2 = 0 $\Rightarrow$ Par(x)

Esta relação é verdadeira para qualquer número, seja ele qual for.

Raciocínio com conhecimento geral

Tendo

- Numero(6)
- Inteiro(6)
- 6 Mod 2 = 0

Então Par(6)

O racicocínio com conhecimento geral procura instanciar esse conhecimento para o nosso caso particular e, usando regras de inferência, produzir a reposta para o problema particular. Aquilo que se pretende salientar é que o conhecimento geral resolve problemas por instanciação. O raciocínio por analogia baseado em casos, não usa instanciação porque não é instanciável. Funciona por semelhança e adaptação.

Conhecimento de casos

Caso 1

Problema

- Heroi(James)
- Vilao(Jaws).
- Pos(James, (2,5)).
- Pos(Jaws, (2, 5)).

Solução

- Disparar(James, Jaws).

Caso 2

Problema

- Heroi(Batman)
- Vilao(Pinguin).
- Pos(Batman, (10,4)).
- Pos(Pinguin, (10, 4)).

Solução

- Disparar(Batman, Pinguin).

Caso 3

Problema

- Heroi(James)
- Vilao(Jaws).
- Pos(James, (2,10)).
- Pos(Jaws, (2, 5)).

Solução

- PassoBaixo(James).

Raciocínio por analogia baseado em casos

Problema:

- Heroi(SpiderMan)
- Vilao(Octopus).
- Pos(SpiderMan, (3,10)).
- Pos(Octopus, (3, 5)).

Qual é a solução deste problema? Isto é, que acção vai fazer o Spider Man?

P: Qual é o caso da base de conhecimentos mais parecido com o problema actual?

R: Caso 3

P: Qual é a analogia entre os dois casos (o novo e o caso 3):

R:

Novo Problema	**Caso 3**
SpiderMan	James
Octopus	Jaws
3	2

i.e., { James/SpiderMan, Jaws/Octupus, 2/3 }

P: Qual a solução do caso 3?

R: PassoBaixo(James)

P: Qual é a solução do caso novo?

R: PassoBaixo(SpiderMan), a qual resulta da aplicação da analogia estabelecida à solução do caso 3

Pode formar-se um novo caso constituído pelo novo problema e a solução derivada por analogia. Se este novo caso for suficientemente diferente dos casos já armazenados deve-se armazena-lo na base de casos do sistema baseado em conhecimento, aumentando assim o conhecimento do sistema. É natural que haja problemas futuros que sejam mais semelhantes ao novo caso do que os outros casos já existentes.

Se o novo caso não for suficientemente diferente dos casos existentes não vale a pena armazená-lo pois a rapidez do sistema diminui com o aumento de casos armazenados.

1 Arquitectura e funcionamento dos sistemas de CBR

Um sistema de CBR armazena os seus casos numa base de conhecimentos a que se chama Base de Casos. Quando surge um problema novo que o sistema tem de resolver, são efectuados os passos de recuperação comparação e selecção, adaptação, e armazenamento.

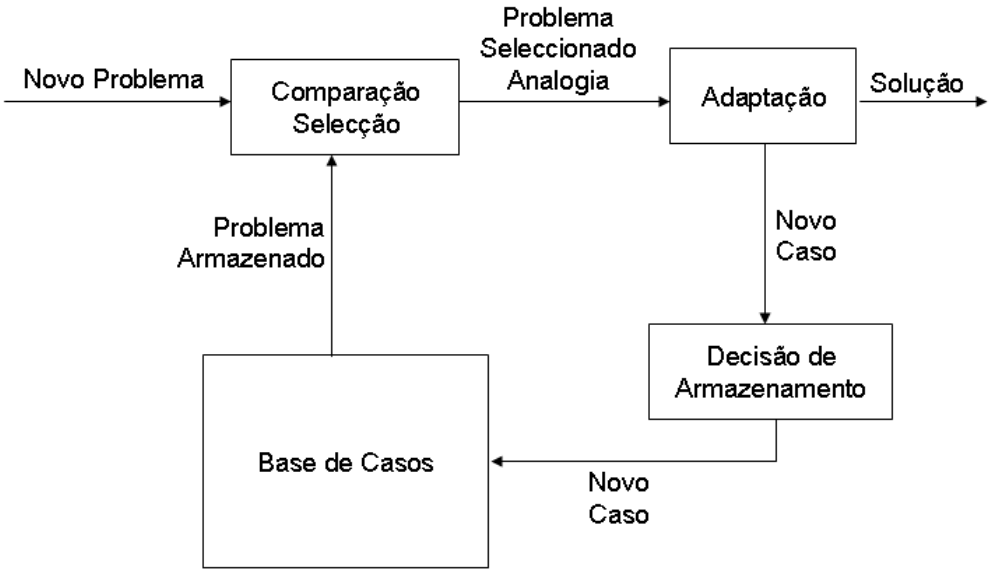

Figura 1 – Arquitectura geral de um sistema de CBR

Na fase de recuperação, comparação e selecção, o sistema consulta a sua base de casos e compara-os com o novo problema. Dessa comparação resultam duas coisas: a analogia entre o problema armazenado na base de casos e o novo problema, e a selecção dos casos mais relevantes para a resolução do novo problema. Nas situações mais simples, o sistema compara os casos existentes com o novo problema e determina uma medida de semelhança entre eles. É seleccionado o caso cuja semelhança com o novo problema for maior.

Na fase de adaptação, o sistema tem de decidir se a solução encontrada no caso seleccionado é adequada à resolução do novo problema. Se for adequada, então o sistema sugere essa solução para o novo problema. Se não for adequada, o sistema tem de adaptar a solução ou soluções dos casos seleccionados ao novo problema. Em geral, a analogia estabelecida entre o novo problema e o caso seleccionado permite adaptar a solução do caso seleccionado à resolução do novo problema.

Por exemplo, se a analogia estabelecida entre os dois casos, especifica que o termo t1.1 do novo problema corresponde ao termo t2.1 do problema armazenado no caso seleccionado, então se a solução do caso seleccionado é S(t2.1), a solução do novo problema será S(t1.1).

Da fase de Adaptação resulta um novo caso, o qual é formado pelo novo problema e pela solução para ele sugerida pelo sistema (quer a solução tenha sido exactamente a solução do caso seleccionado quer tenha resultado de um processo de adaptação).

Na fase de armazenamento, o sistema decide se deve ou não armazenar o novo caso na sua base de casos. Se o novo caso for armazenado, o sistema aprende no sentido em que poderá usar este novo caso para a resolução de situações futuras. Qual o critério usado pelo sistema para decidir se deve ou não armazenar o novo caso na sua base de casos?

A decisão tem de ponderar os benefícios e as desvantagens de armazenar o novo caso. Em termos das desvantagens, o critério mais importante é a perda de eficiência se a base de casos ficar muito grande. Quanto maior for a base de casos, maior será o tempo gasto na fase de recuperação, comparação e selecção.

A vantagem do armazenamento do novo caso é a possibilidade que o sistema passa a ter de usar o novo caso para resolver novos problemas.

O critério mais simples e mais usual para decidir se o caso deve ser armazenado baseia-se na sua semelhança com os casos já existentes na base de casos. Se o novo caso for muito diferente dos casos já armazenados, o sistema decide armazená-lo. Se, pelo contrário, o novo caso for muito semelhante a casos já armazenados, o sistema decide não armazenar o novo caso.

2 Semelhança entre casos e adaptação

Para determinar a semelhança entre dois casos e para poder adaptar a solução de um deles para resolver o problema representado pelo outro, é necessário estabelecer uma analogia entre os dois casos. Isto é, é necessário determinar que elemento de um caso corresponde a cada elemento de outro caso. Por vezes, quando os casos comparados têm dimensões diferentes, é necessário determinar também que elementos de um caso não têm correspondência no outro caso.

As duas secções seguintes discutem a determinação da semelhança entre casos para vários tipos de problema, e várias maneiras de adaptar a solução dos casos seleccionados para encontrar a solução para o novo problema.

2.1 Semelhança entre casos

Quando se compara o problema { Heroi(SpiderMan), Vilao(Octopus), Pos(SpiderMan, (3,10)), Pos(Octopus, (3, 5)) } com o caso 3 { Heroi(James), Vilao(Jaws), Pos(James, (2,10)), Pos(Jaws, (2, 5)) }, com a finalidade de determinar o grau de semelhança e estabelecer a analogia entre eles, estamos a comparar dois conjuntos de termos.

O grau de semelhança e a analogia a que chegamos dependem da escolha dos elementos de um dos conjuntos que comparamos com os elementos do outro conjunto. Por exemplo, se compararmos Heroi(SpiderMan) do primeiro conjunto com Heroi(James) do segundo conjunto, resulta a analogia {James/SpiderMan}. No entanto, se decidirmos comparar Heroi(SpiderMan) com Vilao(Jaws), resulta uma outra analogia {Vilao/Heroi, Jaws/SpiderMan}. Certamente que os valores dos graus de semelhança resultantes serão diferentes nas duas situações.

Quando o sistema de raciocínio baseado em casos compara dois casos, ele não sabe que elemento de um dos conjuntos (i.e., um dos casos) corresponde a cada elemento do outro conjunto (i.e., o outro caso). A maneira de o determinar consiste em efectuar várias comparações entre os conjuntos, uma para cada possível correspondência entre elementos de um e elementos de outro. Para cada comparação efectuada, determina-se o grau de semelhança entre os conjuntos. No final, escolhe-se a correspondência que resulta no grau de semelhança mais elevado.

Exemplo:

C1 = { a1, b1 }

C2 = { a2, b2 }

Alternativa 1		Alternativa 2	
Correspondência	Semelhança	Correspondência	Semelhança
a1 ↔ a2	0.7	a1 ↔ b2	0.9
b1 ↔ b2	0.8	b1 ↔ a2	0.2

Tabela 1 – Correspondências possíveis entre os elementos de C1 e de C2

Usando a correspondência da Alternativa 1 da Tabela 1, o grau de semelhança entre os conjuntos seria 0.75 = (0.7+0.8)/2. Usando a Alternativa 2, o grau de semelhança obtido da mesma forma seria 0.55. Pode concluir-se que a relação de correspondência entre os elementos de C1 e elementos de C2 é a da Alternativa 1, e que a semelhança de C1 e C2 é 0.75.

No exemplo descrito de comparação entre casos, assumimos que se conhecia o grau de semelhança entre cada elemento de um conjunto (i.e., caso 1) e os elementos do outro conjunto (i.e., caso 2). No entanto, em geral, essa semelhança tem de ser calculada. Se os elementos que pretendemos comparar forem termos, então podemos calcular a sua semelhança a partir da semelhança entre os seus functores e das semelhanças entre os seus argumentos.

Por exemplo, se pretendermos determinar a semelhança entre os dois termos f1(a1, b1) e f2(a2, b2), poderemos comparar os functores f1 e f2 e os argumentos a1 / a2 e b1 / b2 e fazer o cálculo.

Será que o peso da semelhança entre functores é igual ao peso da semelhança entre argumentos? Em certos casos, os pesos da semelhança entre functores e da semelhança entre argumentos podem ser os mesmos, mas num caso geral, não é adequado atribuir-lhes pesos idênticos.

Assim sendo, poderá usar-se a seguinte expressão:

Semelhança($\rho_1(\alpha_{1.1}, ..., \alpha_{1.n})$, $\rho_2(\alpha_{2.1}, ..., \alpha_{2.n})$) = $Wf \times$Semelhança(ρ_1, ρ_2) + $Wa_1 \times$Semelhança($\alpha_{1.1}$, $\alpha_{2.1}$) + ... + $Wa_n \times$Semelhança($\alpha_{1.n}$, $\alpha_{2.n}$) em que Wf é o peso da semelhança entre functores para a semelhança entre termos, e Wa_i é o peso da semelhança entre os argumentos i correspondentes.

No exemplo anterior, consideramos que a ordem dos argumentos de um functor é relevante. Nesses casos, o primeiro argumento de um dos termos só pode ser comparado com o primeiro argumento do outro termo. Infelizmente, há casos em que a ordem dos argumentos de um termo é irrelevante. Imaginemos que os termos representam a equivalência entre objectos, por exemplo eq(a1, b1). Tanto faz dizer que a1 é equivalente a b1, como dizer que b1 é equivalente a a1. Nesses casos, a ordem dos argumentos é irrelevante. Consequentemente, ao comparar eq(a1, b1) com eq(a2, b2), a1 pode corresponder tanto a a2 como a b2, e b1 pode corresponder tanto a a2 como a b2. Nestes casos, tem de se calcular a semelhança entre os argumentos de um termo, usando um procedimento idêntico ao descrito para comparar dois conjuntos.

Um sistema de CBR que compara casos que representem conjuntos de características de uma dada situação ou objecto tem de fazer todas as correspondências possíveis entre os elementos dos conjuntos a comparar e, para cada uma dessas correspondências, calcular a medida de semelhança entre os conjuntos. Depois escolhe aquela correspondência (i.e., analogia) para a qual se obtém o maior grau de semelhança. Este processo envolve a geração de permutações dos conjuntos que são representados nos casos.

Em muitas aplicações, os casos armazenados numa base de casos representam descrições de situações ou de objectos. Por exemplo, um caso pode representar a marca e o modelo de um carro, uma anomalia detectada e o diagnóstico dessa anomalia. Neste tipo de problemas, a ordem pela qual são armazenadas as diversas características do objecto ou da situação é totalmente irrelevante. Por exemplo, tanto faz dizer "O carro é da marca VW e seu farol direito da frente não acende" como dizer "O farol direito da frente não acende e a marca do carro é VW".

No entanto, em certos domínios de aplicação, os casos armazenados numa base de casos representam sequências de acções. Num julgamento, é importante saber se a vítima agrediu o réu em primeiro lugar e este ripostou à agressão, ferindo a vítima; ou se o réu foi o primeiro agressor.

Quando os casos representam sequências de acções, a comparação entre o problema actual e um caso passado tem de respeitar a ordem pela qual as acções são executadas nos dois casos comparados. Por exemplo, quando comparamos as sequências de acções [a1.1, a1.2, a1.3] com [a2.1, a2.2, a2.3] não fará sentido tentar estabelecer uma correspondência entre a1.1 e a2.2 e entre a2.1 e a1.2. Quando estão envolvidas sequências de acções, apenas poderemos tentar estabelecer correspondências entre subsequências de um caso e subsequências de outro caso.

A sequência [a, b, c] tem várias subsequências: [a, b, c], [a], [b], [c], [a, b], [a, c], e [b, c]. Assim sendo, ao comparar os casos [a1.1, a1.2, a1.3] com [a2.1, a2.2] podemos ser levados a concluir que a2.1 corresponde a a1.1, que a2.2 corresponde a a1.2 e que falta o elemento correspondente a a1.3. Mas também poderíamos concluir que a2.1 corresponde a a1.2, que a2.2 corresponde a a1.3 e que falta o elemento correspondente a a1.1.

Um sistema de CBR que tenha de comparar casos que representam sequências de acções ou de eventos, terá de tentar fazer todas as correspondências possíveis entre as subsequências de um caso e as subsequências de outro e determinar as medidas de semelhança entre casos relativas a cada uma das possíveis correspondências. Depois escolhe a correspondência (i.e., a analogia) que resultar no maior grau de semelhança.

Há domínios de problemas, em que a comparação de casos se rege por princípios mais fáceis de implementar num sistema computacional. Por exemplo, quando se compara dois objectos com propriedades p1, p2 e p3, só faz sentido comparar o valor de p1 do primeiro objecto com o valor de p1 do segundo objecto, o valor de p2 do primeiro objecto com o valor de p2 do segundo objecto, e o valor de p3 do primeiro objecto com o valor de p3 do segundo objecto. Neste tipo de problemas, o processo

de comparação é mais fácil porque a relação de correspondência entre os elementos de um caso e os elementos do outro caso está previamente especificada.

Por fim, há problemas em que certos elementos de um caso não são comparáveis com certos elementos do outro caso. Por exemplo, não podemos comparar uma intenção com uma crença, pois estas duas entidades mentais são fundamentalmente diferentes, isto é, não são comparáveis. Assim sendo, um sistema de CBR deve saber, para cada domínio de aplicação, que coisas pode comparar quando tenta determinar a analogia e a semelhança entre dois casos.

Nos sistemas de CBR é em geral possível estabelecer um limiar de semelhança que evita que sejam considerados casos cuja semelhança com o novo problema seja muito baixa. Quando o sistema não descobre nenhum caso suficientemente parecido com o problema actual, não é gerada qualquer solução.

2.1.1 Comparação de números e de palavras

A comparação entre os elementos de um caso com os elementos de outros acaba sempre por se reduzir à comparação entre tipos básicos, nomeadamente a comparação entre números e a comparação entre constantes alfanuméricas.

Nos casos mais usuais, a comparação entre números é relativamente simples, embora tenha os seus problemas. Talvez a forma mais adequada seja dizer que a semelhança entre os números x e y é dada pela expressão:

1 - diferença(x, y)

A função *diferença/2* devolve um valor no intervalo [0, 1], em que 0 significa que x e y são absolutamente iguais, e 1 significa que x e y são absolutamente diferentes. Infelizmente não é muito fácil encontrar uma função que se comporte desta forma e que seja aceitável por todas as pessoas que queiram usar o CBR nas suas aplicações.

A comparação de constantes alfabéticas ou alfanuméricas é bem mais problemática. A comparação dos códigos dos caracteres que formam as duas constantes não se pode usar porque não reflecte nenhuma informação relativa ao significado das duas constantes.

Uma solução geralmente apresentada para a comparação de constantes alfanuméricas recorre a hierarquias e redes de conceitos.

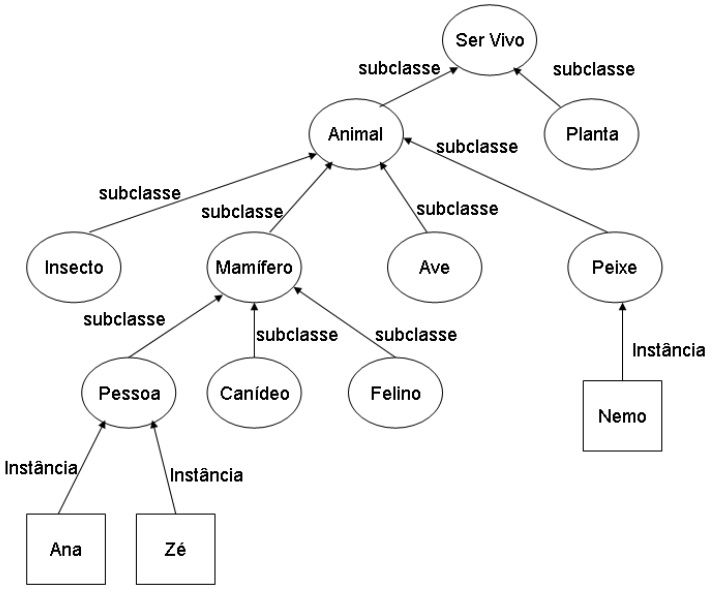

Figura 2 – Hierarquia de conceitos

Usando uma hierarquia de conceitos, é possível desenhar um processo simples que determina a distância entre dois conceitos. A ideia geral consiste em atravessar a porção da hierarquia que liga os dois conceitos um ao outro, contando o número de ligações percorridas. Podemos, por exemplo, admitir que a travessia de um arco contribui com 1 para a distância entre conceitos. Adoptando esta

posição, a distância entre Ana e Zé, na hierarquia representada na Figura 2, seria computada através da seguinte expressão:

$$\text{Distância(Ana, Ze)} = \text{Distância(Zé, Ana)} = 1 + 1 = 2$$

O primeiro contributo 1 resulta de atravessar o arco *Instância* entre *Ana* e *Pessoa*. O segundo contributo 1 resulta de atravessar o arco *Instância* de *Zé* para *Pessoa*, em sentido contrário.

Usando o mesmo processo, a distância entre Ana e Nemo seria dada pela expressão

$$\text{Distância(Ana, Nemo)} = \text{Distância(Nemo, Ana)} = 1 + 1 + 1 + 1 + 1 = 5$$

O processo descrito é muito simples mas tem alguns aspectos criticáveis. Por exemplo, a distância entre Ana e Pessoa (1) é menor do que a distância entre Ana e Zé (2). De alguns pontos de vista, dois indivíduos (instâncias) são mais semelhantes do que um indivíduo e uma classe. No entanto, o processo descrito pode ser alterado para contornar este tipo de problema.

Tendo uma medida da distância não normalizada como a descrita, podemos obter uma nova medida de distância que varia no intervalo [0, 1], dividindo o valor da distância não normalizada, pelo valor da maior distância possível entre os conceitos representados na hierarquia. Tendo uma distância normalizada, podemos determinar a semelhança através de uma expressão idêntica à usada para a semelhança entre números.

O processo descrito para determinar a semelhança entre conceitos representados por constantes alfabéticas depende da existência de uma hierarquia de conceitos como a que se mostra na Figura 2.

Outros domínios haverá em que o cálculo da distância entre constantes alfanuméricas se deveria fazer em redes de conceitos, em vez de hierarquias. E, noutros casos ainda, nem as redes nem as hierarquias de conceitos serão adequadas.

Posto isto não é possível criar uma ferramenta computacional com a possibilidade de determinar a semelhança entre conceitos representados por constantes alfanuméricas, usando técnicas análogas à descrita nesta secção.

2.1.2 O contexto da comparação

A comparação entre os dois mesmos valores poderá dar origem a um valor de semelhança, num determinado contexto, e dar origem a um valor de semelhança diferente, noutro contexto. Por exemplo, a semelhança entre 1 grau centígrado e 2 graus centígrados pode ser muito grande se a escala das temperaturas variar entre -275 e 1000 graus centígrados, mas poderá ser muito pequena se a escala das temperaturas variar entre 0 e 3 graus centígrados.

Num outro exemplo, a comparação entre 1 e 2 pode resultar numa dada medida de semelhança, se 1 e 2 forem euros, mas poderá resultar num valor diferente de semelhança, se 1 e 2 forem pessoas.

Consequentemente, os sistemas de CBR devem permitir que o programador defina as suas funções de comparação de casos, ou de elementos de casos. Na situação mais agradável, o sistema CBR oferece métodos de comparação que poderá usar se o programador estiver contente com eles. Se o programador não estiver contente, poderá programar métodos alternativos.

2.2 Adaptação da solução

Existem vários processos usados na fase de adaptação. O processo mais simples consiste em substituir argumentos dos termos de um caso por argumentos correspondentes (análogos) dos termos do outro caso.

Outro processo, igualmente simples, consiste em substituir os functores dos termos de um caso pelos functores de termos correspondentes do outro caso.

Existe também a possibilidade de combinar a solução proposta por um caso com a solução proposta por outro caso para dar origem à solução para o novo problema.

Além dos processos simples de adaptação referidos, existem outros, mas estes apontamentos quedam-se nestes três métodos.

2.2.1 Substituição de argumentos e de functores

A adaptação por substituição de argumentos é um processo de adaptação em que o sistema CBR substitui um ou mais argumentos no caso seleccionado por argumentos correspondentes do novo problema, para gerar a solução.

A adaptação por substituição de functores é um processo muito semelhante ao da substituição de argumentos. Em vez de se substituírem argumentos, substituem-se functores de termos do caso seleccionado por functores dos termos correspondentes do novo problema.

A adaptação por substituição de argumentos e a adaptação por substituição de functores dependem, de forma muito directa, da analogia estabelecida na fase de recuperação, comparação e selecção de casos.

Ambos os processos recorrem à analogia estabelecida, a qual representa as correspondências entre elementos de um caso e elementos de outro.

Imagine-se que temos o novo problema e o caso seleccionado que se apresentam na Figura 3.

Novo Problema	Caso Seleccionado	
{p(a, b), q(b, f(a))}	Problema	Solução
	{p(a, c), q(c, g(a))}	**g(c)**

Figura 3 – Novo Problema e Caso Seleccionado

Recorrendo à comparação entre o novo problema e o problema do caso seleccionado, podemos admitir que resulta a analogia {c ↔ b, g ↔ f } em que T ↔ Q significa que T corresponde a Q.

No processo de adaptação da solução do caso seleccionado, g(c), para gerar a solução para o novo problema, o sistema de CBR recorre à analogia estabelecida e opera as substituições correspondentes. Substitui g por f e substitui c por b, originando a solução f(b). A substituição de c por b é uma substituição de argumentos. A substituição de g por f é uma substituição de functores.

Estes dois processos de substituição são semelhantes em tudo. No entanto parece razoável supor que uma substituição de functores é um passo mais arriscado do que uma substituição de argumentos. Basta imaginar que f poderá ser *matar*, enquanto que g poderá ser *beijar*. Será lícito o sistema apresentar uma solução para um problema em que, às cegas, substitui *beijar* por *matar*?

Se for desejado que a diferença entre functores tenha um impacto maior no raciocínio de um sistema CBR, isso pode ser conseguido durante a comparação dos problemas. Por exemplo, se for impensável substituir o functor *beijar* pelo functor *matar*, basta que a comparação entre *beijar* e *matar* dê origem a uma semelhança nula (ou quase). Assim, um caso com esta diferença em relação ao novo problema nunca chega a ser seleccionado pois não ultrapassará o limiar de comparação.

2.2.2 Composição

A adaptação por composição é um processo de adaptação em que as soluções de dois ou mais casos são combinadas para produzir a solução para o novo problema. Para haver adaptação por composição é pois necessário que o passo de recuperação, comparação e selecção possa seleccionar mais que um caso para construir a solução. Além disso, existem muitas maneiras de combinar soluções.

Por exemplo, pode fazer-se uma média dos valores propostos na solução; ou então, pode juntar-se as soluções dos dois casos seleccionados para obter a solução do novo problema.

Imaginemos que usamos um sistema CBR para controlar o movimento de um robot, o qual dispõe da acção *seguir_em_frente(X)* em que *X* é uma medida em metros. Se a solução de um dos casos seleccionados é *seguir_em_frente(30)*, e a solução do outro caso seleccionado é a acção *seguir_em_frente(40)*, possivelmente é razoável fazer uma média dos valores 30 e 40 para obter a solução para o problema: *seguir_em_frente(35)*.

Imaginemos agora, numa outra situação, que a solução de um dos casos seleccionados é "virar na primeira à direita"; e que a solução do outro dos casos seleccionados é "entrar na primeira porta". Talvez a solução para o novo problema possa ser "virar na primeira à direita e entrar na primeira

porta". Mas também poderia ser "entrar na primeira porta e virar na primeira à direita". Pode até acontecer que nenhuma destas soluções seja adequada para o novo problema.

A substituição por composição é um assunto relativamente complexo que não pode ser apresentado com rigor no tempo destinado a esta matéria nas aulas de TSI.

3 Sistema *Prolog Case*

Esta secção descreve sucintamente o sistema *Prolog Case*, um sistema de CBR implementado em Prolog. O *Prolog Case* será usado nas aulas práticas de TSI. O funcionamento interno do *Prolog Case* não é apresentado em detalhe nesta secção. Apenas serão descritos os aspectos relevantes à sua utilização para a implementação de sistemas baseados em conhecimento que usem o raciocínio baseado em casos como método de representação e de raciocínio.

3.1 Utilização do Prolog Case

Para utilizar o *Prolog Case*, é necessário carregar os seus ficheiros no interpretador de Prolog. Tendo carregado os ficheiros do *Prolog Case*, podem efectuar-se as seguintes tarefas:

- Criar e armazenar uma base de casos;
- Parametrizar o sistema; e
- Utilizar o raciocinador *Prolog Case* para obter soluções para problemas.

Os casos de uma base de casos são representados através do predicado *case/2*. O primeiro argumento do predicado case/2 é a descrição de um problema. O segundo argumento é a solução para esse problema.

O *Prolog Case* tem de ser configurado através de um conjunto de parâmetros a serem definidos pelo programador:

- *comparison_threshold*: Limiar de semelhança a partir do qual o sistema considera que um dado caso pode fornecer a solução para o novo problema.
- *storage_threshold:* Limiar de semelhança até ao qual o sistema decide guardar o novo caso que resulta da junção do novo problema com a solução sugerida para o sistema.
- *problem_matching_method*: Método a ser usado pelo *Prolog Case* para comparar o novo problema com o problema armazenado num caso. Podem usar-se três métodos de comparação: a comparação por emparelhamento directo (chamada *match*, no *Prolog Case*), a comparação envolvendo permutações dos elementos dos conjuntos que caracterizam os problemas a comparar, e a comparação de subsequências das sequências que descrevem as situações a comparar.

Também é necessário especificar que termos são comparáveis, qual é o método usado na comparação dos seus argumentos, e que valor deve ser usado para multiplicar o resultado da comparação dos argumentos de modo a obter o resultado da comparação entre os termos. Estas especificações fazem-se usando os predicados *top_comparable/5* e *comparable_args/8*.

O *Prolog Case* disponibiliza o predicado *cbr/3* para efectuar raciocínio baseado em casos. O *Prolog Case* efectua um tipo de raciocino em que a solução proposta para o novo problema pode resultar da adaptação da solução proposta no caso mais semelhante ao novo problema, existente na base de casos. O primeiro argumento do predicado *cbr/3* é a descrição do novo problema. Este argumento tem obrigatoriamente que ser fornecido ao sistema. O segundo argumento é a solução para o novo problema. Em geral, este argumento será uma variável não instanciada que receberá a sugestão do sistema. Finalmente, o terceiro argumento é o valor da semelhança entre o novo problema e o problema armazenado no caso seleccionado para solucionar o novo problema. Em geral, este argumento será uma variável não instanciada que receberá o valor da semelhança.

O predicado *cbr/3* pode ser usado em qualquer aplicação em Prolog.

As secções seguintes especificam mais detalhes sobre estas três actividades com o *Prolog Case*.

3.2 Distribuição e carregamento de ficheiros

O *Prolog Case* é um programa implementado em LPA Prolog por Bruno Gonçalves, um elemento do laboratório de investigação *"We, the Body, and the Mind"* da ADETTI, um centro de estudos associado ao departamento de Ciências e Tecnologias da Informação do ISCTE.

Sendo um programa em Prolog, a sua utilização passa pelo carregamento, no interpretador Win-Prolog da LPA, dos ficheiros que o constituem:

- cbr.pl
- general_purpose.pl
- match_term_set_comparison.pl
- permutation_term_set_comparison.pl
- solution_adaptation.pl
- subsequence_term_set_comparison.pl
- term_set_comparison.pl
- term_comparison_score.pl

Para carregar os ficheiros do *Prolog Case* no interpretador basta recorrer ao predicado *ensure_loaded/1* e carregar o ficheiro cbr.pl, o qual se encarrega de carregar todos os outros ficheiros necessários. Para isso, num dos ficheiros que constituem o sistema CBR que pretendemos implementar, é necessário usar a directiva

:- ensure_loaded(*Pathname do ficheiro CBR.pl*).

3.3 Criar e armazenar uma base de casos

Os casos, no *Prolog Case*, são guardados em ficheiro e carregados no interpretador de Prolog. Cada caso é armazenado num facto que recorre ao predicado *case/2*.

Em vez de representar cada caso num facto *case/2*, usando a base de dados interna do Prolog, o programador é livre de definir o predicado *case/2* de outra forma qualquer, em particular, pode definir *case/2* como um predicado de acesso a uma base de dados relacional que armazenaria os casos de forma permanente.

O predicado *case/2* tem dois argumentos: o primeiro é a descrição de um problema; o segundo argumento é a solução desse problema:

% case(*Problema*, *Solução*)

Tanto a descrição do problema como a solução são representados por listas de termos Prolog. Cada termo do problema ou da solução pode representar seja o que for. Por exemplo, tanto o problema como a solução podem ser conjuntos de proposições, os quais representam implicitamente a sua conjunção. Neste caso, representa-se apenas conhecimento puramente declarativo. Cada caso capta uma relação entre uma dada situação ou objecto descritos por um conjunto de proposições (a descrição do problema) e uma conclusão adequada também ela representada através de um conjunto de proposições (a solução do problema). Este tipo de representação pode ser usado, por exemplo, em sistemas de diagnóstico de equipamento ou de pessoas. As proposições representadas do lado do problema descrevem os dados e os sintomas do equipamento ou da pessoa; as proposições do lado da solução descrevem o diagnóstico do equipamento ou da pessoa.

Alternativamente, o problema pode ser um conjunto de proposições que representa a sua conjunção, enquanto que a solução pode ser uma lista de acções que representa, por exemplo, a sua sequência. Este tipo de casos serve, por exemplo, para representar conhecimento de controlo de acção. Basicamente, um caso representa a acção a efectuar numa situação descrita pelas proposições que fazem parte da descrição do problema. É um método de representação e raciocínio adequado para controlar robots, agentes de software ou outro tipo de sistemas.

Também se pode optar por ter uma base de casos em que o problema é uma sequência de acções ou de eventos, enquanto que a solução é uma conjunção de proposições. Este tipo de representação é adequado, por exemplo, para problemas de interpretação ou de classificação de situações dinâmicas correspondentes a sequências de eventos ou de acções. É um tipo de representação útil por exemplo para sistemas de apoio a decisões judiciais. A sequência de acções ou de eventos representada no lado do problema descreve uma dada história ou sequência de acontecimentos relativa a um crime; a solução poderá ser a sentença do juiz relativamente à pena aplicada ao condenado.

E podemos até representar casos mistos em que, por exemplo, o primeiro elemento da lista que representa a solução é o diagnóstico e o segundo elemento é a acção que resolve o problema descrito e diagnosticado.

A flexibilidade, no sentido de permitir captar uma grande variedade de tipos de conhecimento, foi a principal razão que suporta o esquema de representação de casos usado no *Prolog Case*.

Como ilustração da representação de casos, através do predicado *case/2*, no *Prolog Case*, vamos supor que pretendemos uma base de casos que pode ser usada para sugerir a acção ou as acções a efectuar em situações em que pretendemos obter um determinado artigo científico (ou outro documento qualquer).

Cada caso representa um problema que corresponde à descrição de uma situação em que se pretende ter acesso a um dado artigo; e uma solução que corresponde a uma ou mais acções que poderiam ser executadas para ter acesso ao artigo desejado, na situação descrita.

A descrição do problema deve especificar o artigo a que se pretende aceder, deve captar relações entre esse artigo e outras entidades relevantes ao problema, por exemplo entre o artigo e o seu autor, e entre o autor e a instituição onde trabalha, ou entre o autor e o seu endereço de correio electrónico.

A Figura 4 mostra alguns dos casos armazenados na base de casos.

```
case([ artigo_desejado('Learning through observation'),
       autor('Learning through observation', jaçanã),
       paginaWeb(jaçanã, 'http://iscte.pt/~jacana/')],

     [ abrir_pagina('http://iscte.pt/~jacana/') ]).

case([ artigo_desejado('Agent based simulation'),
       autor('Agent based Simulation', bruno),
       telefone(bruno, 960860360) ],

      [ telefonar(960860360) ]).

case([ artigo_desejado('Brainstorm assistance tool'),
       autor('Brainstorm assistance tool', antonio),
       paginaWeb(antonio, 'http://www.we-b-mind.org/~tony/'),
       telefone(antonio, 931111333) ],

      [ abrir_pagina('http://www.we-b-mind.org/~tony/') ]).
```

Figura 4 – Base de casos

A base de casos apresentada na Figura 4 capta conhecimento sobre situações específicas ou episódios. O primeiro caso representa que, na situação específica em que o artigo desejado se chama *'Learning through observation'* escrito pela *Jaçanã* cuja página na Web é *'http://iscte.pt/~jacana/'*, a acção adequada (para ter acesso ao artigo) é abrir a página na Web *'http://iscte.pt/~jacana/'*.

O segundo caso diz que, na situação específica em que o artigo desejado se chama *'Agent based simulation'*, escrito pelo *Bruno* cujo telefone é *960860360*, a acção adequada (para obter o artigo) é telefonar para o número *960860360*. Assume-se que não existe a página na Web do Bruno (ou que o sistema não a conhece).

O terceiro caso diz que, na situação em que se pretende um artigo chamado *'Brainstorm assistance tool'* escrito pelo António, o qual tem telefone número *931111333* e página na Web *'http://www.we-b-mind.org/~tony'*, a acção adequada (para obter o artigo) é abrir a página *'http://www.we-b-mind.org/~tony'*.

É importante salientar novamente que este é o conhecimento explicitamente representado nos casos. Ele diz respeito apenas a situações, objectos e pessoas específicas. A utilização que o raciocínio CBR faz deste conhecimento é outra coisa, é algo adicional ao conhecimento explicitamente representado.

3.4 Parametrizar o sistema

A configuração do sistema *Prolog Case* tem duas partes. Na primeira, estabelecem-se os valores de alguns parâmetros do sistema. Na segunda parte, especifica-se que termos são comparáveis e como é que essa comparação é efectuada. A Figura 5 mostra um exemplo de configuração para o sistema de CBR cuja base de casos se encontra ilustrada na Figura 4.

```
% Parâmetros

comparison_threshold(0.5).
storage_threshold(0.7).
problem_matching_method(permutations).

% Termos comparáveis

/* top_comparable(
     ComparableId, ExistingFunctorSpec, NewFunctorSpec,
     ArgsComparisonMethod, Similarity) */

% Functores dos termos comparáveis dos problemas

top_comparable(1, artigo_desejado/1, artigo_desejado/1, match, 1).

top_comparable(2, autor/2, autor/2, match, 1).

top_comparable(3, paginaWeb/2, paginaWeb/2, match, 1).

top_comparable(4, telefone/2, telefone/2, match, 1).


% Argumentos comparáveis


/* comparable_args(
     ComparableId, ContextId,
     ExistingTermIndex, NewTermIndex, Arg1, Arg2, ComparisonMethod,
     Similarity) */

% Artigo

comparable_args(5, 1, 1, 1, Artigo1, Artigo2, no_comparison, 1):-
  artigo(Artigo1), artigo(artigo2), Artigo1 \= Artigo2.

comparable_args(6, 2, 1, 1, Artigo1, Artigo2, no_comparison, 1):-
  artigo(Artigo1), artigo(artigo2), Artigo1 \= Artigo2.

% Autor

comparable_args(7, 2, 2, 2, Autor1, Autor2, no_comparison, 1):-
  pessoa(Autor1), pessoa(Autor2), Autor1 \= Autor2.

comparable_args(8, 3, 1, 1, Autor1, Autor2, no_comparison, 1):-
  pessoa(Autor1), pessoa(Autor2), Autor1 \= Autor2.

comparable_args(9, 4, 1, 1, Autor1, Autor2, no_comparison, 1):-
  pessoa(Autor1), pessoa(Autor2), Autor1 \= Autor2.

% URL

comparable_args(10, 3, 2, 2, Url1, Url2, no_comparison, 1):-
  url(Url1), url(Url2), Url1 \= Url2.


artigo(Artigo):- atom(Artigo).

pessoa(Autor):-  atom(Autor).

url(Url):-
  atom(Url).      % Isto poderia ser por exemplo verificar se tem a sintaxe de um URL
```

Figura 5 – Configuração do *Prolog Case*

As primeiras três linhas de configuração mostradas na Figura 5 estabelecem os valores dos parâmetros limiar de comparação (*comparison_threshold*), limiar de armazenamento (*storage_threshold*), e

método de comparação usado para efectuar a comparação entre os elementos do problema novo e os elementos dos casos armazenados (*problem_matching_method*).

O limiar de comparação (*comparison_threshold*) igual a 0.5 significa que, para encontrar a solução para o novo problema, o *Prolog Case* apenas considera casos com uma semelhança igual a ou maior que 0.5 com o novo problema. O limiar de comparação pode variar entre 0 e 1, inclusive.

O limiar de armazenamento (*storage_threshold*) igual a 0.7 significa que o *Prolog Case* apenas armazena novos casos se a semelhança entre o caso seleccionado e o novo problema for inferior a 0.7 (e superior ao limiar de comparação estabelecido por *comparison_threshold*). O limiar de armazenamento pode variar entre 0 e 1, inclusive.

Finalmente, o *Prolog Case* dispõe de três métodos de comparação: *match*, *subsequences* e *permutations*. Quando *problem_matching_method* é *match*, isso significa que, quando o *Prolog Case* compara a lista de termos que representa o novo problema com a lista de termos que representa o problema de um caso armazenado, o termo de ordem i do novo problema só pode ser comparado com o termo de ordem i do problema armazenado.

O método de comparação *subsequences* é usado quando se pretende comparar duas sequências sem obrigar a que as comparações se façam exclusivamente entre elementos com o mesmo número de ordem na sequência. Imaginemos que pretendemos comparar as duas sequências S1 : [t1.1, t1.2, t1.3] e S2 : [t2.1, t2.2], usando o método *subsequences*. Para efectuar a comparação, usando este método, fazem-se as correspondências apresentadas na Tabela 2.

Analogia 1	**Analogia 2**	**Analogia 3**
• T1.1 corresponde a T2.1 • T1.2 corresponde a T2.2 • Em S2, falta o elemento correspondente a T1.3	• T1.1 corresponde a T2.1 • Em S2, falta o elemento correspondente a T1.2 • T1.3 corresponde a T2.2	• Em S2, falta o elemento correspondente a T1.1 • T1.2 corresponde a T2.1 • T1.3 corresponde a T2.2

Tabela 2 – Analogias entre duas sequências

O *Prolog Case* determina a semelhança entre as duas sequências, para cada uma das três possibilidades apresentadas na Tabela 2, e admite que a correspondência (analogia) correcta é aquela para a qual o valor da semelhança é mais elevado. O método *subsequences* é computacionalmente muito mais exigente do que o método *match*, mas menos exigente que o método *permutations*.

O método de comparação *permutations* é usado para comparar dois conjuntos, sempre que não for possível impor uma correspondência conhecida à partida entre os elementos de um conjunto e os elementos do outro conjunto. Este processo de comparação envolve a geração de permutações nos conjuntos de modo a encontrar todas as possíveis correspondências entre eles. Em geral, usando este método geram-se muito mais hipóteses de correspondências do que usando o método *subsequences*. Admitindo que pretendemos comparar os conjuntos S1 : {t1.1, t1.2, t1.3} e S2 : {t2.1, t2.2}, usando o método *permutations*, obtêm-se as possíveis alternativas apresentadas na Tabela 3.

Analogia 1	**Analogia 2**	**Analogia 3**
• T1.1 corresponde a T2.1 • T1.2 corresponde a T2.2 • Em S2, falta o elemento correspondente a T1.3	• T1.1 corresponde a T2.1 • T1.3 corresponde a T2.2 • Em S2, falta o elemento correspondente a T1.2	• T1.2 corresponde a T2.1 • T1.1 corresponde a T2.2 • Em S2, falta o elemento correspondente a T1.3.
Analogia 4	**Analogia 5**	**Analogia 6**
• T1.2 corresponde a T2.1 • T1.3 corresponde a T2.2 • Em S2, falta o elemento correspondente a T1.1	• T1.3 corresponde a T2.1 • T1.1 corresponde a T2.2 • Em S2, falta o elemento correspondente a T1.2	• T1.3 corresponde a T2.1 • T1.2 corresponde a T2.2 • Em S2, falta o elemento correspondente a T1.1

Tabela 3 – Analogias entre dois conjuntos

O *Prolog Case* determina a semelhança entre os dois conjuntos para todas as relações de analogia alternativas e admite que a analogia correcta é aquela que corresponde ao valor mais elevado de semelhança.

O método de comparação por permutações é computacionalmente muito dispendioso mas o *Prolog Case* usa um método mais expedito que evita que, na realidade, sejam geradas e avaliadas todas as permutações. Apesar de mais expedito, o método usado pelo *Prolog Case* garante que se obtém sempre o mesmo resultado que seria obtido através da geração de todas as permutações. Isto é, apesar de mais expedito, o método usado garante uma solução óptima. Infelizmente, este método mais espedito tem, em alguns problemas, de gerar de facto todas as permutações.

Para mais detalhes sobre o método realmente utilizado pelo *Prolog Case*, pode consultar-se a tese de mestrado de Bruno Gonçalves [Gonçalves 2006].

Além da configuração dos parâmetros descritos, a Figura 5 mostra também a especificação dos termos que são comparáveis e do método de comparação que deve ser usado para comparar os seus argumentos. Para isso são usados dois predicados: *top_comparable/5* que serve para indicar a semelhança entre os functores dos termos comparáveis dos problemas e o método de comparação a usar para comparar os argumentos desses functores; e *comparable_args/8* que serve para especificar a semelhança entre os argumentos comparáveis dos termos comparáveis.

Uma cláusula top_*comparable(ComparableId, ExistingFunctorSpec, NewFunctorSpec, ArgsComparisonMethod, Similarity)* estabelece um contexto de comparação identificado por *ComparableId*, e significa que os termos de casos já existentes formados pelo functor especificado por *ExistingFunctorSpec* e os termos de um problema novo formados pelo functor especificado em *NewFunctorSpec* são comparáveis, que a semelhança entre os functores especificados é *Similarity*, e que os argumentos dos termos com os functores especificados são comparáveis através do método *ArgsComparisonMethod*. Por exemplo, *top_comparable(2, autor/2, autor/2, match, 1)* estabelece o contexto de comparação 2, e significa que, nesse contexto, um termo de um problema de um caso existente de aridade 2 formado a partir do functor *autor* é comparável com um termo de um problema novo com aridade dois formado pelo functor *autor*; que as respectivas colecções de argumentos devem ser comparadas através do método *match* de comparação e que a semelhança entre os dois functores é 1. Esta semelhança entre functores é multiplicada pela semelhança entre argumentos para determinar a semelhança entre os termos. O método de comparação *match* (já previamente explicado) significa que o argumento i de um termo só pode ser comparado com o argumento i do outro termo. Além de *match*, existem ainda os métodos de comparação *subsequences* e *permutations*, também eles já explicados.

Cada cláusula *top_comparable/5* é identificada univocamente por um identificador único. Actualmente, a unicidade desse identificador é da inteira responsabilidade do programador. Este identificador estabelece um contexto de comparação que pode ser referido mais tarde em factos *comparable_args/8*.

A cláusula *comparable_args(ComparableId, ContextId, ExistingArgIndex, NewArgIndex, ExistingArg, NewArg, ComparisonMethod, Similarity)* estabelece o contexto de comparação identificado por *ComparableId*, e significa que a semelhança entre o argumento *ExistingArg* (i.e., argumento número *ExistingArgIndex* de um termo de um problema de um caso existente, no contexto de comparação *ContextId*) e o argumento *NewArg* (i.e., argumento número *NewArgIndex* de um termo do novo problema, no contexto de comparação *ContextId*) é *Similarity*. Por exemplo, A cláusula

```
comparable_args(7, 2, 2, 2, Autor1, Autor2, no_comparison, 1):-
  pessoa(Autor1), pessoa(Autor2), Autor1 \= Autor2.
```

significa que, no contexto de comparação nº 2, o qual poderia ter sido estabelecido por uma cláusula top_*comparable/5* ou *comparable_args/8* (no nosso exemplo, o contexto nº 2 é o contexto de comparação de dois termos *autor/2*, o qual foi criado por uma cláusula *top_comparable/5*), o segundo argumento (2) de um termo de um problema de um caso já existente pode ser comparado com o segundo argumento (2) de um termo de um novo problema desde que esses argumentos sejam pessoas (*pessoa(Autor1)* e *pessoa(Autor2)*) e sejam diferentes um do outro. Se forem iguais, o sistema usa o método de comparação por omissão. Neste contexto e nestas circunstâncias, a semelhança entre os valores dos segundos argumentos dos dois termos é 1. Finalmente, o método de comparação *no_comparison* indica que os argumentos não têm argumentos ou, se os tiverem, a semelhança não depende deles.

Cada cláusula *comparable_args/8* é identificada univocamente por um identificador único. Actualmente, a unicidade desse identificador é da inteira responsabilidade do programador. O primeiro argumento do predicado *comparable_args/8* é o identificador da cláusula. Este identificador estabelece

um contexto de comparação que pode ser referido mais tarde noutros factos *comparable_args/8*. O segundo argumento de *comparable_args/8* especifica o contexto em que se diz que dois argumentos são comparáveis. Este argumento (*ContextId*) tem de ser o identificador de outra cláusula *top_comparable/5* ou *comparable_args/8*. Os identificadores das cláusulas *top_comparable/5* têm que ser diferentes dos identificadores das cláusulas *comparable_args/8*, porque todos eles estabelecem contextos de comparação diferentes.

O *Prolog Case* permite especificar termos com aridades indeterminadas. Para isso, o valor da aridade a especificar num dado termo deve ter o valor n, por exemplo *occurred/n*.

Este processo de configurar a comparação entre dois termos é muito flexível pois permite a utilização de métodos de comparação diferentes para encontrar a semelhança entre as colecções de argumentos dos dois termos comparados.

Além disso, evita que o sistema tente comparar termos não comparáveis, o que permite ganhos notáveis de eficiência.

3.5 Utilizar o raciocinador

O *Prolog Case* efectua um tipo de raciocínio CBR relativamente simples. Na fase de recuperação, comparação e selecção é seleccionado, no máximo, um caso da base de casos: é o caso mais semelhante ao novo problema, apenas se a medida de semelhança for superior ao limiar especificado. Como é seleccionado apenas um caso, a fase de adaptação é também bastante simplificada pois não efectua composição das soluções dos casos seleccionados para obter a solução para o novo problema. Actualmente, na fase de adaptação, o *Prolog Case* apenas substitui argumentos e functores na solução do caso seleccionado com os functores e os argumentos correspondentes do novo problema.

O *Prolog Case* disponibiliza o predicado *cbr/3* para fazer raciocínio baseado em casos. O predicado *cbr/3* tem três argumentos: o novo problema, a solução e a medida da semelhança entre o caso seleccionado da base de casos para derivar a solução para o novo problema.

O novo problema é especificado através de uma lista de termos. Tal como acontece para o predicado *case/2*, os termos contidos na especificação de um problema podem representar seja o que for, por exemplo proposições, acções, objectos. No entanto, é forçoso que o significado da descrição do novo problema se obtenha da mesma forma que o significado da descrição de problemas dos casos armazenados na base de casos do sistema. O novo problema tem de ser fornecido quando o predicado *cbr/3* é usado.

A solução é em geral uma variável não instanciada que recebe a solução sugerida pelo *Prolog Case* para o novo problema. Se a solução for dada ao sistema, ela é comparada com a solução produzida pelo sistema e o resultado dessa comparação é o resultado do predicado *cbr/3*.

A semelhança é em geral uma variável não instanciada que recebe a semelhança entre o caso seleccionado pelo sistema para gerar a solução e o novo problema. Se a semelhança for dada ao sistema, ela é comparada com a semelhança que o sistema produziria. Se fossem iguais, o predicado *cbr/3* teria sucesso, caso contrário teria insucesso.

Usando o predicado *cbr/3*, a base de casos da Figura 4 e a parâmetrização da Figura 5, podem efectuar-se diversas perguntas ao *Prolog Case*. A Figura 6 mostra algumas interacções com o *Prolog Case*.

```
?- cbr([artigo_desejado('The use of plans'),
        autor('The use of plans', martha),
        paginaWeb(martha, 'http://www.eecs.umich.edu/~pollackm/')],
       Accao, Semelhanca).

Accao = [abrir_pagina('http://www.eecs.umich.edu/~pollackm/')],
Semelhança = 1
yes

?- cbr([artigo_desejado('The Role of Emotions'),
        autor('The Role of Emotions', paolo),
        paginaWeb(paolo, 'http://www.ofai.at/~paolo.petta/'),
        telefone(paolo, 777999555)], Accao,  Semelhanca).

Accao = [arbir_pagina('http://www.ofai.at/~paolo.petta/')] ,
Semelhança = 0.875

?- cbr([artigo_desejado('A caça aos gambusinos'),
        autor('A caça aos gambusinos', pedro),
        telefone(pedro, 395564456)], Accao,  Semelhanca).

Accao = [telefonar(395564456)],
Semelhança = 0.833333333333333
yes
```

Figura 6 – Interacções com o sistema CBR

Na primeira interacção, pretende saber-se que acção efectuar na situação em que o artigo desejado se chama *"The use of plans"* e foi escrito por *Martha Pollack* cuja página na Web é *"http://www.eecs.umich.edu/~pollackm/"*.

O *Prolog Case* percorre a sua base de casos (Figura 4) e selecciona o caso mais semelhante ao problema recebido. Trata-se do primeiro caso da base de casos. Como a semelhança entre o problema recebido e o caso seleccionado (1) é maior que o limiar de comparação (0.5, Figura 5), o *Prolog Case* usa o caso seleccionado para adaptação.

A adaptação recorre à analogia estabelecida entre o caso seleccionado e o novo problema posto ao sistema. Usando os métodos de comparação especificados na configuração do sistema (Figura 5), resulta a seguinte analogia

"Learning through observation" corresponde a *"The use of plans"*

"Jaçanã" corresponde a *"Martha"*

"http://iscte.pt/~jacana/" corresponde a "*http://www.eecs.umich.edu/~pollackm/"*

Usando esta analogia, a solução prevista no caso seleccionado (abrir_pagina('*http://iscte.pt/~jacana/*')) é adaptada substituindo "*http://iscte.pt/~jacana/*" por "*http://www.eecs.umich.edu/~pollackm/"*, dando origem à solução sugerida pelo sistema:

[abrir_pagina('http://www.eecs.umich.edu/~pollackm/')]

A segunda e a terceira interacções seguem um raciocínio semelhante. A diferença é que, na segunda interacção, foi seleccionado o terceiro caso para gerar a resposta; e na terceira interacção, foi seleccionado o segundo caso para produzir a resposta. Salienta-se que, nas interacções descritas, mais que um caso da base de casos emparelha com o novo problema (com uma dada semelhança) mas, em cada uma delas, apenas o caso mais semelhante é seleccionado para gerar a solução.

Na Figura 6, o predicado *cbr/3* foi directamente usado no interpretador Prolog. No entanto, como acontece com qualquer predicado Prolog, o *cbr/3* pode ser usado no seio de outro programa. Em particular, pode escrever-se um programa que use lógica vaga para determinar a semelhança entre casos num sistema CBR.

4 Referências Bibliográficas

[Gonçalves 2006] Gonçalves, B. 2006. "Motivação Artificial: Objectivos, Interrupções e Esforço". Tese de Mestrado de Engenharia Informática e de Telecomunicações. Departamento de Ciências e Tecnologias da Informação do ISCTE.

[Kolodner 1993] Kolodner, J. 1993. "Case-Based Reasoning". Morgan Kaufmann Publishers, San Mateo, CA

[Veloso 1997] Veloso, M. 1997. "Merge strategies for multiple case plan replay". In Case-Based Reasoning Research and Development, Proceedings of ICCBR-97, the Second International Conference on Case-Based Reasoning, pages 413-424. Springer Verlag

[Veloso and Carbonell 1993] Veloso, M. and Carbonell, J. 1993. "Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization". *Machine Learning*, 10:249-278

[Veloso, Mulvehill and Cox 1997] Veloso, M.; Mulvehill, A.; and Cox, M. 1997. "Rationale-supported mixed-initiative case-based planning" In *Proceedings of IAAI-97, Innovative Applications of Artificial Intelligence*, pages 1072-1077, Providence, Rhode Island, July 1997