Tecnologia para Sistemas Inteligentes

Apontamentos para as aulas sobre

Representação e Raciocínio com Conceitos Imprecisos: Sistemas de Regras Baseadas na Lógica Vaga

Luís Miguel Botelho

Departamento de Ciências e Tecnologias da Informação do ISCTE-IUL

Abril de 2017

Tecnologias para Sistemas Inteligentes

Apontamentos para as aulas

Índice

Representação e Raciocínio com Conceitos Imprecisos: Sistemas de Regras Baseadas na Lógica Vaga

O objetivo deste capítulo é o de explicar as bases mínimas formais e o de dar suporte mais aplicado que permitam o desenho e desenvolvimento de sistemas baseados em conhecimento que usam regras e raciocínio baseados na lógica vaga.

Algum do nosso conhecimento é preciso e pode ser encarado como se fosse absolutamente certo. No entanto, talvez sem nos darmos conta, a maior parte do conhecimento humano envolve imprecisão e incerteza. Embora a imprecisão possa ser vista como um caso particular de incerteza, quando consideramos métodos de representação de conhecimento e de raciocínio para o desenvolvimento de sistemas baseados em conhecimento, é útil diferenciar com toda a clareza aquilo que pretendemos dizer com o termo "incerteza" daquilo que pretendemos dizer com a palavra "imprecisão".

Vimos, a propósito da representação de regras condição conclusão com fator de confiança, que usamos o termo incerteza para situações em que podemos não confiar absolutamente na informação ou no conhecimento armazenado numa base de conhecimentos de um sistema baseado em conhecimento. Por exemplo, se a informação armazenada numa base de conhecimentos for proveniente de um canal de informação que introduza ruído e distorção, é natural que pretendamos associar um certo grau de desconfiança relativamente à qualidade / rigor dessa informação. Por exemplo, se um conjunto de regras com o objetivo de modelar o comportamento de uma dada bolsa de valores nos for dado por uma pessoa não especialista nesse assunto, temos o direito e até a obrigação de associar a cada uma dessas regras um certo grau de incerteza para expressar a nossa atitude de certa desconfiança em relação a esse corpo de conhecimento.

O conceito imprecisão, tal como ele é usado neste texto, não representa nenhuma espécie de desconfiança quanto à qualidade e rigor da informação e do conhecimento usado. O termo imprecisão é usado aqui para caracterizar conceitos cuja definição não pode ser precisa / rigorosa, se usarmos os métodos clássicos de representação tais como a lógica matemática e os conjuntos clássicos. Por exemplo, a altura a partir da qual se pode considerar que um Português é alto não tem uma definição precisa. Certamente que um Português de 2 metros será sem sobra de dúvidas um Português alto. Também é certo que um Português de 1.50m não é seguramente um português alto. Mas será que um Português de 1.75m pode ser considerado alto? Gostaríamos de poder dizer que um Português de 1.75m é alto em certa medida mas não será realmente alto. Também poderíamos dizer que um Português de 1.75m de altura corresponde bem a um Português de altura média.

Um outro exemplo seria o conceito de temperatura ambiente elevada, em Portugal. Se tentarmos estabelecer 25ºC como o limiar de temperatura a partir do qual se consideraria uma temperatura ambiente elevada em Portugal teríamos problemas de bom senso. Se 25ºC é elevado, então o que dizer de 24,9ºC? E se 25ºC é uma temperatura elevada, não haverá maneira de dizer que a temperatura 28ºC é também elevada mas em maior grau?

E se falarmos de velocidade automóvel numa autoestrada? O problema é certamente o mesmo.

Usando uma abordagem baseada na teoria clássica de conjuntos, não temos uma maneira razoável de resolver estes problemas. Se decidirmos que o conjunto das alturas dos portugueses altos é formado por todas as alturas iguais ou superiores a 1.80m, então 1.80m e 2m pertencem, tanto um como outro, ao conjunto; e 1.79 não pertencerá a esse conjunto. E o mesmo se passa em relação à temperatura ambiente em Portugal e em relação à velocidade automóvel numa autoestrada. Se decidirmos que o conjunto das temperaturas ambientes elevadas é formado por todas as temperaturas iguais ou superiores a 25ºC, então 25ºC e 30ºC pertencem, tanto uma como outra ao conjunto; e 24.9ºC não pertencerá a esse conjunto.

A teoria de conjuntos vagos ("*fuzzy sets*") permite representar muito melhor todos estes tipos de conceitos vagos; e a lógica vaga (*"fuzzy logic"*) serve de base à representação e ao raciocínio com regras condição/conclusão que envolvem conceitos imprecisos como os de que falamos.

Para melhor clarificar a diferença entre conhecimento incerto e conhecimento que envolve conceitos imprecisos, mostra-se, na Figura 1, três regras. A primeira ilustra um exemplo de conhecimento incerto mas que envolve apenas conceitos totalmente precisos. A segunda representa conhecimento

absolutamente certo, no sentido em que temos a certeza da sua validade, mas que envolve conceitos imprecisos. A terceira e última regra representa conhecimento simultaneamente incerto e que envolve imprecisão.

Se a Patricia Highsmith é a única autora de um livro L e a Maria dos Anjos não tem esse livro então L é selecionado para oferecer à Maria dos Anjos
Se a temperatura ambiente é elevada e a humidade relativa é muito baixa então o tempo de evaporação da água é mais curto do que se a temperatura ambiente fosse baixa e a humidade relativa fosse muito elevada
Se um livro é de *suspense* então a Maria dos Anjos gosta desse livro

Figura 1 – Imprecisão e incerteza

A primeira regra envolve apenas conceitos bem precisos. Uma pessoa ou é autor ou não é, especialmente se nos estivermos a referir a autor único. Ter um livro é igualmente um conceito bem preciso (não considerando casos mais raros de possuir a meias ou de partilhar). É igualmente fácil atribuir um significado rigoroso ao conceito de selecionar um livro para oferecer. Basta dizer que ele significa que o incluímos no conjunto dos livros do qual escolheremos a nossa oferta. Patricia Highsmith, Maria dos Anjos e livro são igualmente precisos (se, no domínio considerado, os nomes das pessoas as puderem identificar). No entanto, a regra não reflete conhecimento absolutamente seguro. É bem possível que a Maria dos Anjos não possua algum livro da Patricia Highsmith mas do qual não goste e que, por isso, não devesse ser selecionado para lhe ser oferecido. Há muitos mais fatores na escolha de um livro para além do autor.

A segunda regra reflete conhecimento absolutamente certo. Em circunstâncias ambientais normais, é absolutamente certo que a água evapore mais depressa se a temperatura ambiente for elevada e se a humidade relativa for baixa do que nas condições opostas. Apesar de termos a certeza de que a regra é válida, ela envolve vários conceitos imprecisos. Temperatura elevada, temperatura baixa, humidade muito baixa e humidade muito alta são conceitos cuja definição precisa, em termos da teoria clássica de conjuntos, não pode ser representada com razoabilidade. Temperatura elevada, temperatura baixa, humidade elevada e humidade baixa não têm definição precisa pelas mesmas razões já apontadas anteriormente relativamente a altura e temperatura. Além disso, ainda temos o qualificador "muito" que modifica os adjetivos baixa e elevada. Quando é que uma humidade deixa de ser apenas elevada e passa a ser muito elevada?

Finalmente, a terceira regra é incerta. É muito possível que a Maria dos Anjos não goste de todos os livros de *suspense*. Além disso, os conceitos "livro de *suspense*" e "gostar de um livro" são certamente muito imprecisos.

Consideremos agora que pretendemos criar um sistema baseado em conhecimento com regras baseadas na lógica vaga, por exemplo, um sistema para determinar o valor da gratificação a dar ao empregado, num restaurante. Poderíamos querer representar uma regra como "Se a comida é deliciosa e o serviço é excelente, então a gratificação é generosa". A lógica vaga (*Fuzzy Logic*) e os sistemas de representação que usam regras baseadas em lógica vaga permitem representar conhecimento como este, envolvendo conceitos imprecisos. Em termos computacionais, esta regra pode escrever-se da seguinte maneira[1]:

IF food IS Delicious AND service IS Excellent THEN tip IS Generous

Nesta regra, as palavras em maiúsculas indicam operadores específicos dos sistemas baseados em lógica vaga; as palavras "food", "service" e "tip" são variáveis vagas, e as palavras "Delicious", "Excellent" e "Generous" são valores vagos, os quais são representados através de conjuntos vagos.

Num sistema baseado em lógica vaga, como este, é necessário representar as regras vagas, é necessário definir os conjuntos vagos que servem de valores das variáveis, e é ainda necessário efetuar um sem número de escolhas que falaremos mais tarde.

Assim como a lógica de predicados de primeira ordem assenta na teoria dos conjuntos, também a lógica vaga assenta na teoria dos conjuntos vagos. Este capítulo da matéria inclui uma introdução aos

[1] A sinaxe exata da regra depende da ferramenta computacional usada; esta é uma mera possibilidade. Veremos, por exemplo, que a sintaxe do FUZ-is, ferramenta usada na cadeira, não é exatamente esta.

conjuntos vagos, uma introdução à lógica vaga e a análise da representação através de regras vagas e do mecanismo de raciocínio usado nos sistemas baseados em lógica vaga.

1 Conjuntos Vagos

Na teoria clássica dos conjuntos, dado um determinado conjunto, qualquer objeto pertence a esse conjunto ou então não lhe pertence. Nos conjuntos vagos, qualquer objeto pertence a qualquer conjunto com um determinado grau de pertença representado por um valor real entre 0 e 1. 1 significa que o objeto pertence absolutamente ao conjunto; e 0 significa que o objeto não pertence de facto ao conjunto. Dito de outra forma, um grau de pertença com valor contínuo entre 0 e 1 generaliza o conceito clássico de pertença.

Um conjunto vago define-se em relação a um universo de discurso (i.e., um domínio), através de uma função de pertença. A função de pertença de um conjunto vago faz corresponder a cada elemento do universo de discurso um valor no intervalo [0,1], o qual representa o grau de pertença desse elemento do universo de discurso no conjunto vago.

Para tornar o assunto mais concreto, voltemos à regra vaga apresentada no início deste componente da bibliografia, a qual se reproduz aqui para conveniência do leitor:

IF food IS Delicious AND service IS Excellent THEN tip IS Generous

Uma variável vaga toma um conjunto vago como valor. Ao conjunto dos vários conjuntos vagos que uma variável vaga pode tomar chama-se quantização vaga da variável. Os conjuntos vagos que formam a quantização de uma variável vaga são todos definidos para o mesmo universo de discurso. Na regra apresentada, é possível que a variável vaga "food" possa tomar os valores "Lousy", "Medium", e "Delicious".

Apresenta-se agora a definição dos conjuntos vagos referidos na regra e de outros relacionados com eles. Os conjuntos vagos que representam os valores vagos que a variável "food" pode tomar são todos definidos para o universo de discurso (i.e., o domínio de variação) da qualidade da comida. Podemos classificar a qualidade da comida numa escala de 0 a 9, em que 0 significa a comida de pior qualidade e 9 significa a comida de melhor qualidade. A Figura 2 representa graficamente os conjuntos "Lousy", "Medium", e "Delicious".

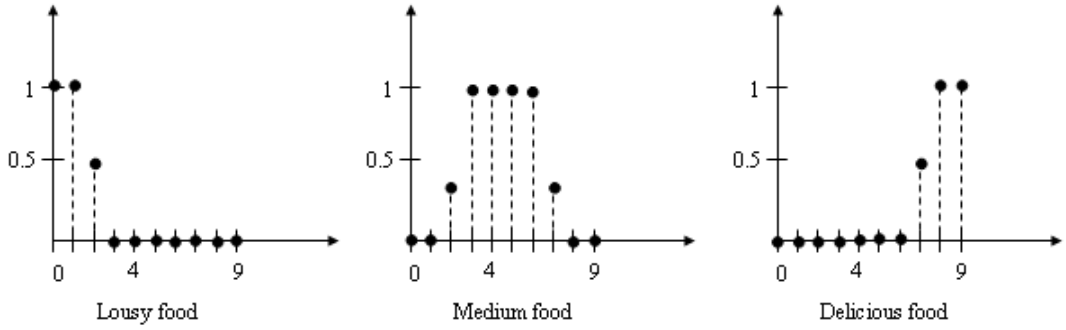


Figura 2 – Conjuntos vagos "Lousy", "Medium", e "Delicious"

A comida má ("Lousy") é representada pelo conjunto vago "Lousy". Uma comida com qualidade 0 é definitiva e absolutamente uma comida má, por isso, o valor 0 de qualidade da comida pertence em grau 1 ao conjunto "Lousy". Se a comida tem qualidade 1, então também pode ser considerada má comida, por isso a qualidade 1 pertence igualmente com grau 1 ao conjunto "Lousy". Uma comida com qualidade 2 já não é tão má assim. Vamos dizer que uma comida com qualidade 2 já começa a não ser má. Assim sendo, a qualidade 2 pertence ao conjunto "Lousy", mas apenas com grau de pertença 0.5. As comidas com qualidade superior a 2 deixam em absoluto de ser más, por isso as qualidades 3 a 9 pertencem ao conjunto "Lousy" com grau 0 (i.e., não pertencem). Os graus de pertença usados nas definições deste e dos próximos conjuntos são da inteira responsabilidade de quem define os conjuntos. Não há nenhuma regra nem teoria que se possa aplicar para determinar os graus de pertença.

A comida com qualidade 3, 4, 5 ou 6 é uma comida definitivamente média. Assim as qualidades 3, 4, 5 e 6 pertencem em grau 1 ao conjunto "Medium". A comida com qualidade 2 já não é propriamente uma

comida média mas também ainda não é definitivamente má. Assim, decidiu-se que a qualidade 2 pertence apenas com grau 0.3 ao conjunto "Medium". A qualidade 7 já é bem melhor do que média, por isso pertence ao conjunto "Medium" com grau de pertença 0.3. Finalmente, todas as outras qualidades (0, 1, 8 e 9) já não são consideradas qualidades médias. Apesar de termos escolhido 0.3 como grau de pertença da qualidade 2 e da qualidade 7 ao conjunto "Medium", poderíamos igualmente ter escolhido graus de pertença diferentes para as duas qualidades. Nada nos obriga a definir conjuntos de forma simétrica.

Quanto ao conjunto "Delicious", ele contem as qualidades mais elevadas em grau absoluto. As qualidades 8 e 9 pertencem-lhe com grau de pertença 1. A comida com qualidade 7 já começa a ser deliciosa mas não totalmente, por isso a qualidade 7 pertence ao conjunto "Delicious" com grau de pertença 0.5. Todos as outras não têm qualidade suficiente para serem deliciosas pelo que pertencem a "Delicious" em grau 0.

As explicações apresentadas e a representação gráfica da Figura 2 podem representar-se formalmente da seguinte maneira.

Lousy = {0/1, 1/1, 2/0.5, 3/0, 4/0, 5/0, 6/0, 7/0, 8/0, 9/0}

Medium = {0/0, 1/0, 2/0.3, 3/1, 4/1, 5/1, 6/0.3, 7/0, 8/0, 9/0}

Delicious = {0/0, 1/0, 2/0, 3/0, 4/0, 5/0, 6/0, 7/0.5, 8/1, 9/1}

Nesta maneira formal de representar conjuntos vagos, cada conjunto vago é representado através de um conjunto de pares (elemento / grau de pertença). O grau de pertença de um elemento num dado conjunto é dado por uma função de pertença, geralmente representada pela letra grega μ indexada pelo nome do conjunto. Por exemplo, a função de pertença relativa ao conjunto vago "Lousy" será geralmente representada por μ_{Lousy}, a qual tem a seguinte definição (tendo em conta a proposta apresentada): $\mu_{Lousy}(x) = 1$ para $x \in \{0, 1\}$; $\mu_{Lousy}(2) = 0.5$; e $\mu_{Lousy}(x) = 0$ para $x \in \{3, 4, 5, 6, 7, 8, 9\}$. A função de pertença de um conjunto vago define-se apenas no universo de discurso desse conjunto.

Para que se possa usar a regra do exemplo, seria ainda necessário definir os conjuntos relativos a um serviço excelente e a uma gratificação generosa. À semelhança da qualidade da comida, a qualidade do serviço num restaurante pode ser classificada numa escala de 0 (mínimo) a 9 (máximo). Nesse cenário, o conjunto que representa um serviço excelente seria {0/0, 1/0, 2/0, 3/0, 4/0, 5/0, 6/0, 7/0.5, 8/1, 9/1}, o qual tem a representação gráfica da Figura 3.

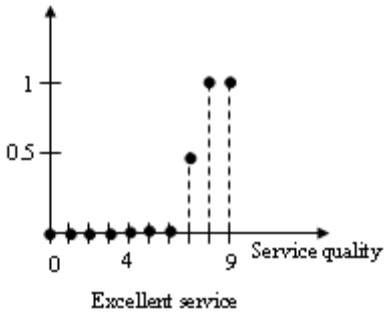


Figura 3 – Excellent Service

A gratificação poderá variar entre não dar gratificação e 10% do valor da refeição. Vamos considerar os valores de gratificações em percentagem 0, 5, 10%. Para este universo de variação da gratificação, poderemos definir o conjunto que representa uma gratificação generosa como um conjunto vago, para o qual, a gratificação 10% pertence em grau 1, e todas as outras gratificações têm grau de pertença 0. Formalmente, o conjunto vago que representa uma gratificação generosa fica definido da seguinte maneira:

Generous = {0/0, 5/0, 10/1}

"Generous" aparece também representado graficamente na Figura 4.

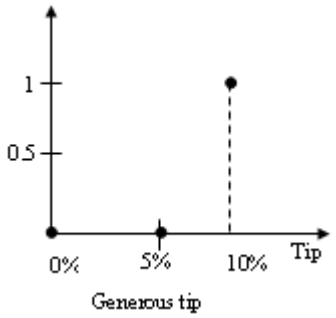


Figura 4 – Generous Tip

O conjunto "Generous" é um caso particular de um conjunto vago que representa um valor exato. Neste caso, representa o valor exato 10%. Este exemplo mostra com mais clareza que os conjuntos vagos podem também representar valores exatos (*"crisp value"*, na terminologia dos conjunto vagos e da lógica vaga).

Embora tenhamos apresentado conjuntos vagos através de representações discretas, é muito comum apresentá-los através de funções contínuas. Em geral, recorre-se a dois tipos de funções contínuas (ou prolongáveis por continuidade): as compostas por troços de reta, e as formadas por linhas curvas. A Figura 5 mostra duas representações alternativas, usando funções contínuas, do conjunto vago "medium" exibido na Figura 2.

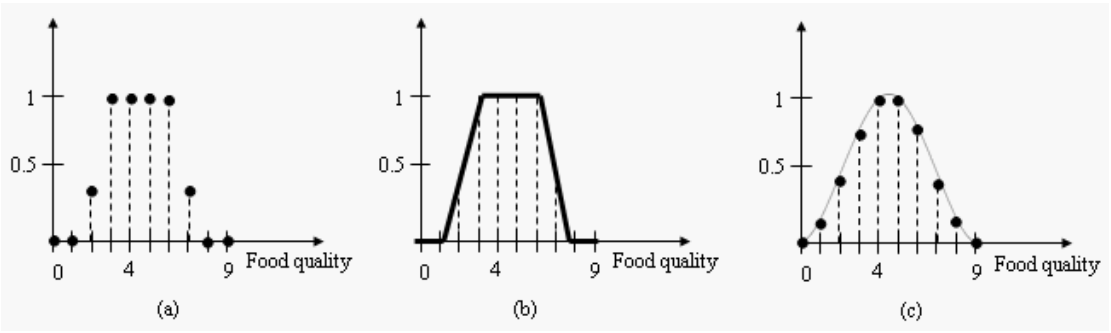


Figura 5 – Representações Alternativas do Conjunto "Medium"

A Figura 5 (a) apresenta a representação discreta original do conjunto "Medium". As figuras (b) e (c) são duas representações contínuas do mesmo conjunto. (b) é formada a partir de troços de reta, e (c) é formada por linhas curvas. A subsecção 1.1 enumera vários exemplos de conceitos imprecisos e suas possíveis representações através de conjuntos vagos.

1.1 Exemplos de representação de conceitos imprecisos

Exemplos com base na Altura e no Peso

Universo de discurso da altura: 140, 150, 160, ... cm

Variável valga: *Altura*

Valores (conjuntos) vagos (i.e., quantização vaga de *Altura*): baixo, médio, alto

Universo de discurso do peso (adultos): 40, 45, 50, 55, 60, 65, ... Kg

Variável vaga: *Peso*

Valores (conjuntos) vagos (i.e., quantização vaga de *Peso*): leve, médio, pesado

Exemplos com base na quantidade de cigarros e no risco de cancro

Universo de discurso (nº de cigarros por dia): 0, 2, 4, 6, 8, 10, 20 cigarros

Variável vaga: Quantidade de tabaco por dia

Valores (conjuntos) vagos (i.e., quantização vaga de *Quantidade de tabaco*): pouco, médio, muito

Universo de discurso referente ao Risco de Cancro: 0, 1, 2, 3, 4, 5 unidades de risco

Variável vaga: *Risco de Cancro*

Valores (conjuntos) vagos (i.e., quantização vaga de *Risco de cancro*): baixo, normal, elevado

Exemplos com base no tempo de travagem

Universo de discurso: 2, 3, 4, 5, 6, 7, 8, 9, 10 segundos

Variável vaga: Tempo de resposta dos travões

Valores (conjuntos) vagos (i.e., quantização vaga de *Tempo de resposta*): rápido, médio, lento

Exemplos com base no pêndulo invertido

Universo de discurso: -50, -40, -30, -20, -10, 0, 10, 20, 30, 40, 50 Newton

Variável vaga: *Força a aplicar*

Valores (conjuntos) vagos (i.e., quantização vaga de *Força*): grande negativa, pequena negativa, zero, pequena positiva, grande positiva.

Apesar de quase todos exemplos apresentados de variáveis vagas possuírem apenas três possíveis valores vagos (e.g., baixo, médio, alto), isso não é nenhuma obrigação. São igualmente muito usadas variáveis com cinco possíveis valores vagos (e.g., muito baixo, baixo, médio, alto e muito alto). Há também outros casos, como o da variável Força com valores vagos grande negativa, pequena negativa, zero, pequena positiva e grande positiva. Não há nenhuma limitação ao número de valores vagos que uma variável vaga pode tomar.

1.2 Operações e relações com conjuntos vagos

À semelhança do que se passa nos conjuntos clássicos, existem diversas operações relativas a conjuntos vagos.

União e intersecção de conjuntos vagos

$\mu_{A\cup B}(u) = \text{Max}(\mu_A(u), \mu_B(u))$

$\mu_{A\cap B}(u) = \text{Min}(\mu_A(u), \mu_B(u))$

Isto significa que, se um objeto u pertence a um conjunto A com grau de pertença $\mu_A(u)$, e pertence a um conjunto B com grau de pertença $\mu_B(u)$, então pertence à reunião de A com B, com grau de pertença $\text{Max}(\mu_A(u), \mu_B(u))$, e pertence à intersecção de A com B com um grau de pertença $\text{Min}(\mu_A(u), \mu_B(u))$.

Complemento de um conjunto vago

O complemento de um conjunto vago S define-se no mesmo universo de discurso que S. O grau de pertença de x ao complemento de S é $1-\mu(x)$.

Igualdade de dois conjuntos vagos

Dois conjuntos vagos A e B são iguais sse as suas funções de pertença forem iguais.

Subconjuntos

$A \subseteq B$ sse $\mu_A(u) \leq \mu_B(u)$ para todo o u pertencente ao universo de discurso

Produto, soma e diferença algébricos

O produto algébrico de dois conjuntos vagos A e B, com funções de pertença μ_A e μ_B respetivamente, é o conjunto cuja função de pertença é $\mu_{A\times B}(x) = \mu_A(x)\times\mu_B(x)$ para todo o x pertencente ao universo de discurso.

A soma algébrica de dois conjuntos vagos A e B com funções de pertença μ_A e μ_B respetivamente, é o conjunto cuja função de pertença é $\mu_{A+B}(x)= \mu_A(x)+\mu_B(x)$ para todo o x pertencente ao universo de discurso.

A diferença algébrica de dois conjuntos vagos A e B com funções de pertença μ_A e μ_B respetivamente, é o conjunto cuja função de pertença é $\mu_{A-B}(x)=\mu_A(x)-\mu_B(x)$ para todo o x pertencente ao universo de discurso.

Produto, soma e diferença limitados

O produto limitado de dois conjuntos vagos A e B com funções de pertença μ_A e μ_B respetivamente, é o conjunto com função de pertença $Max(0, \mu_A(x)+\mu_B(x)-1)$ para todo o x pertencente ao universo de discurso.

A soma limitada de dois conjuntos vagos A e B com funções de pertença μ_A e μ_B respetivamente, é o conjunto cuja função de pertença é $Min(1, \mu_A(x)+\mu_B(x))$ para todo o x pertencente ao universo de discurso.

A diferença limitada de dois conjuntos vagos A e B com funções de pertença μ_A e μ_B respetivamente, é o conjunto cuja função de pertença é $Max(0, \mu_A(x)-\mu_B(x))$ para todo o x pertencente ao universo de discurso.

Normalização de um conjunto: NORM(A)

$\mu_{NORM(A)}(u) = \mu_A(u) / MAX(\{\mu_A(u): u \in U\})$ em que U é o universo de discurso de A (e de NORM(A)), e em que MAX é uma função que devolve o maior elemento de um conjunto.

1.3 Propriedades interessantes

Os conjuntos vagos não exibem algumas das propriedades mais intuitivas dos conjuntos clássicos. Destas, as mais relevantes são:

A reunião de um conjunto com o seu complementar é um conjunto cuja função de pertença não é forçosamente igual a 1 para todos os elementos do universo de discurso. A Figura 6 mostra um conjunto vago S e o seu complementar. Pode ver-se que a reunião de S com o seu complementar tem uma função de pertença não constante.

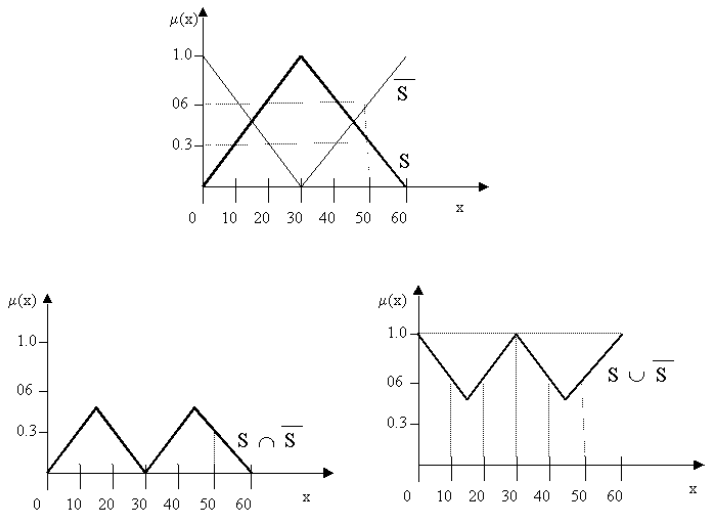


Figura 6 Intersecção e reunião de um conjunto com o seu complementar

A intersecção de um conjunto vago com o seu complementar é um conjunto vago cuja função de pertença não é forçosamente igual a 0 para todos os elementos do universo de discurso (Figura 6).

1.4 "Hedges" ou modificadores

Os modificadores são operadores que podem ser aplicados a um conjunto vago ou a um valor *"crisp"* (i.e., valor exato) e produzem um conjunto vago. Os modificadores são importantes na medida em que permitem criar novos conjuntos vagos sem ter que os representar explicitamente. Entre os modificadores mais conhecidos, destacam-se a *concentração*, a *diluição* e o operador *cerca de* ou *aproximadamente*.

Concentração

A concentração2 é uma operação que se aplica a um conjunto vago para obter outro conjunto vago com uma função de pertença mais "concentrada". Sendo S um conjunto vago com função de pertença μS, a concentração de S é um conjunto vago com função de pertença μCONC(S)(x)= μS(x)2.

Diluição

A diluição[3] é uma operação que se aplica a um conjunto vago para obter outro conjunto vago com uma função de pertença mais "diluída", isto é, mais alargada. Sendo S um conjunto vago com função de pertença μ_S, a diluição de S é um conjunto vago com função de pertença $\mu_{DIL(S)}(x)= \mu_S(x)^{½}$.

Cerca de, ou aproximadamente

"Cerca de" ou *"aproximadamente"* é um operador que se aplica a um valor exato (*"crisp"*) para obter um conjunto vago com uma função de pertença em forma de sino centrada em torno do valor exato, por exemplo *"cerca de 500"*. "Cerca de 500" é um conjunto vago em forma de sino com grau de pertença igual a 1 em 500.

Conjunto complementar

O complementar de um conjunto pode ser considerado um modificador que se aplica a um conjunto vago para obter outro conjunto vago cuja função de pertença é o complemento para 1 da função de pertença do conjunto original.

1.5 Funções de pertença usuais

Várias funções de pertença têm sido muito usadas em diversas aplicações de conjuntos vagos e de lógica vaga. De entre estas, destacam-se a função trapezoidal, a função triangular, a função sigmoide, a função Z, e a função em forma de sino.

Seguidamente apresentam-se figuras com a representação destas funções de pertença.

---

[2] O modificador de concentração é frequentemente referido pela designação “muito”, por exemplo “muito alto”. Infelizmente, esta designação tão comum parece-nos desajustada porque colide com a definição frequente de conjuntos vagos com o prefixo muito, como por exemplo muitoAlto, que são via de regra muito diferentes do resultado da aplicação do modificador de concentração.

[3] O modificador de diluição é frequentemente referido pela designação “pouco”, por exemplo “pouco dispendioso”. Infelizmente, esta designação tão comum parece-nos desajustada porque colide com a definição frequente de conjuntos vagos com o prefixo pouco, como por exemplo poucoDispendioso, que são via de regra muito diferentes do resultado da aplicação do modificador de diluição.

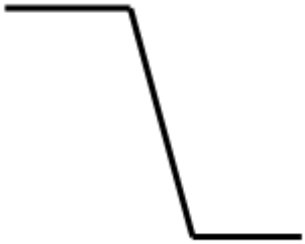
		
(a) Sino (Π)	Sigmoide (Função S)	Simétrica da Sigmoide
		
Trapezoide	Função Simétrica da Z	Função Z
		
Triângulo		

Tabela 1 – Algumas Funções de Pertença Típicas

Alguns estudos indicam que as aplicações de conjuntos vagos (especialmente em sistemas computacionais baseados na lógica vaga) não dependem excessivamente de detalhes da forma das funções de pertença usadas. É quase indiferente ter uma função trapezoidal ou uma função em forma de sino. O mais importante são o seu formato geral e as suas dimensões.

2 Lógica Vaga

Assim como a semântica da lógica de predicados de primeira ordem é baseada em conjuntos (o significado de um predicado é um conjunto), a semântica da lógica vaga recorre a conjuntos vagos. As proposições da lógica vaga, em vez de terem um valor lógico pertencente ao conjunto {verdade, falso} têm um grau de verdade no intervalo contínuo [0,1].

A proposição mais simples em lógica vaga tem o formato (x is A) em que x é uma variável vaga e A é conceito impreciso, representado por um conjunto vago, por exemplo (temperatura is TemperaturaElevada). O grau de verdade da proposição (x is A) é dado pela função de pertença dos elementos do universo de discurso de x no conjunto vago A. Para clarificar este conceito vamos recorrer ao exemplo da gratificação num restaurante.

Vejamos em que medida é que a proposição (gratificação is GratificaçãoElevada) é verdadeira. Vamos supor que a gratificação pode tomar os valores exatos 0, 5%, e 10% do valor da conta. Diz-se que o universo de discurso da gratificação é G={0, 5, 10}. Temos de definir o conjunto vago que representa o conceito impreciso *GratificaçãoElevada*. Neste exemplo, admitimos que uma gratificação de 0 não é elevada, que uma gratificação de 5% é apenas muito pouco elevada, e que uma gratificação de 10% é realmente uma gratificação elevada. Tendo em conta estas ideias, o conjunto *GratificaçãoElevada* pode

ser definido da seguinte forma: GratificaçãoElevada={0/0, 5/0.2, 10/1}. Usando a letra g para designar os valores do universo de discurso G, esta representação significa que a função de pertença deste conjunto, $\mu_{GratificaçãoElevada}(g)$ toma o valor 0 quando g=0, toma o valor 0.2 quando g=5 e toma o valor 1 quando g=10.

De acordo com a regra semântica relativa ao grau de verdade das proposições atómicas, o grau de verdade da proposição (gratificação is GratificaçãoElevada) é dada pela função de pertença do conjunto *GratificaçãoElevada*, isto é de $\mu_{GratificaçãoElevada}$. Isto significa que para o valor exato da gratificação g=0, a proposição (gratificação is GratificaçãoElevada) é verdade em grau 0 (i.e., não é verdade); para uma gratificação com valor exato g=5, a proposição (gratificação is GratificaçãoElevada) é verdade apenas em grau 0.2; e para uma gratificação com valor exato g=10, a proposição (gratificação is GratificaçãoElevada) é verdade em grau máximo (em grau 1).

Vamos agora formalizar esta definição que acabamos de descrever e explicar com um exemplo. Se designarmos a proposição atómica (x is A) por P, se x e A forem definidos no universo de discurso U a cujos elementos chamamos u, e se a função de pertença do conjunto vago A for $\mu_A(u)$, então a medida em que P é verdade, $\theta_P : U \rightarrow [0, 1]$, é dada pela expressão $\theta_P(u) = \mu_A(u)$. Neste texto, enquanto as funções de pertença são indicadas pela letra μ, as funções de verdade são indicadas pela letra θ.

As proposições vagas compostas obtêm-se pela combinação de proposições simples usando as conectivas vagas correspondentes às conectivas usuais da lógica de primeira ordem, por exemplo a negação, a conjunção, a disjunção e a implicação vagas.

O grau de verdade de uma proposição vaga composta resulta da combinação dos graus de verdade das proposições mais simples que a constituem. O grau de verdade de uma proposição composta é calculado com base nos graus de verdade das proposições que a constituem, através de um processo empírico. Diversos autores propuseram regras de cálculo diferentes para determinar os graus de verdade de proposições compostas.

Sendo P ≡ (x is A) e Q ≡ (y is B) duas proposições vagas, em que x e A se definem no universo U cujos valores são denotados pela letra u, e em que y e B se definem no universo V cujos valores são denotados pela letra v, as funções de verdade da conjunção, da disjunção e da negação podem ser as seguintes:

Função de verdade da conjunção, usando o método dos mínimos: $\theta_{P\wedge Q}(u, v) = Min(\theta_P(u), \theta_Q(v))$

Função de verdade da disjunção, usando o método dos máximos: $\theta_{P\vee Q}(u, v) = Max(\theta_P(u), \theta_Q(v))$

Função de verdade da negação, usando o método do complemento: $\theta_{\neg P}(u) = 1-\theta_P(u)$

As funções de verdade apresentadas são apenas propostas (aliás, bem disseminadas) que resultam essencialmente da experiência dos seus autores ou dos seus utilizadores. No entanto, é possível encontrar propostas diferentes para se determinar o grau em que a conjunção e a disjunção são verdadeiras. Para a conjunção, além do método dos mínimos, pode usar-se o método do produto, e o método do produto limitado. Para a disjunção, além do método dos máximos, existe o probor (*probabilistic or*) e a soma limitada. Quanto à negação, há maior consenso.

Função de verdade da conjunção, usando o método do produto: $\theta_{P\wedge Q}(u, v) = \theta_P(u) \times \theta_Q(v)$

Função de verdade da disjunção, usando o método do Probor: $\theta_{P\vee Q}(u, v) = Probor(\theta_P(u), \theta_Q(v))$

em que $Probor(a, b) = a + b – a \times b$.

Função de verdade da conjunção, usando o produto limitado: $\theta_{P\wedge Q}(u, v) = Max(\theta_P(u)+\theta_Q(v)-1, 0)$

Função de verdade da disjunção, usando a soma limitada: $\theta_{P\vee Q}(u, v) = Min(\theta_P(u)+\theta_Q(v), 1)$

Para clarificar este conceito, vamos recorrer ao exemplo da conjunção (temperatura is TemperaturaElevada) $\wedge$ (humidade is HumidadeBaixa), em que o universo da temperatura é T={10, 20, 30, 40, 50}, cujos elementos serão denotados pela letra t, e o universo da humidade é H={30, 50, 70, 90} (em percentagens), cujos elementos serão denotados pela letra h.

Admitamos as seguintes definições dos conjunto vagos que representam os conceitos imprecisos *temperatura elevada* e *humidade baixa*:

TemperaturaElevada={10/0, 20/0, 30/0.2, 40/1, 50/1}

HumidadeBaixa={30/1, 50/0.2, 70/0, 90/0}

As definições apresentadas correspondem a funções de pertença do conjunto *TemperaturaElevada* e *HumidadeBaixa* de acordo com as duas tabelas seguintes:

t	10	20	30	40	50
$\mu_{TemperaturaElevada}(t)$	0	0	0.2	1	1

h	30	50	70	90
$\mu_{HumidadeBaixa}(h)$	1	0.2	0	0

O grau de verdade de P≡(temperatura is TemperaturaElevada) é dado pela função de pertença do conjunto TemperaturaElevada, isto é, $\theta_P = \mu_{TemperaturaElevada}$. O grau de verdade da proposição Q≡(humidade is HumidadeBaixa) é dado pela função de pertença do conjunto HumidadeBaixa, isto é $\theta_Q=\mu_{HumidadeBaixa}$. O que diz a regra semântica relativa à conjunção é, se usarmos o método dos mínimos, que o grau de verdade da conjunção é o menor dos graus de verdade de cada uma das proposições que constituem a conjunção. No nosso caso, temos que o grau de verdade de (temperatura is TemperaturaElevada) ∧ (humidade is HumidadeBaixa) é uma função dos valores da temperatura (t) e da humidade (h), $\theta_{P\wedge Q}(t, h) = Min(\theta_P(t), \theta_Q(h)) = Min(\mu_{TemperaturaElevada}(t), \mu_{HumidadeBaixa}(h))$. Vejamos melhor o que isso significa. Por exemplo, se o valor exato da temperatura for t=30º e o valor exato da humidade for h=70%, $\theta_{P\wedge Q}(30, 70) = Min(\theta_P(30), \theta_Q(70)) = Min(0.2, 0)=0$. Isto é, a proposição (temperatura is TemperaturaElevada) ∧ (humidade is HumidadeBaixa) tem um grau de verdade 0 se o valor exato da temperatura for 30º (pouco elevada) e se o valor exato da humidade for 70% (bastante elevada), o que é compreensível pois a humidade deveria ser baixa. No entanto, se a temperatura for 30º e a humidade for 50% (já mais baixa, mas não muito), o grau de verdade da conjunção (temperatura is TemperaturaElevada) ∧ (humidade is HumidadeBaixa) é $\theta_{P\wedge Q}(30, 50) = Min(\theta_P(30), \theta_Q(50)) = Min(0.2, 0.2)=0.2$. Isto é, a conjunção é já um pouco verdade (0.2) porque a temperatura é um pouco alta, e a humidade é um pouco baixa.

A função de verdade de uma conjunção de duas proposições envolvendo variáveis vagas diferentes (como no caso do exemplo precedente) pode ser representada numa tabela de duas entradas: uma para os valores de um dos universos de discurso, e outra para os valores do outro universo de discurso. Cada entrada da tabela (correspondente a um dado par de valores dos universos de discurso) indica o grau de verdade da conjunção.

A implicação vaga é a conectiva mais interessante da lógica vaga, no sentido em que permite fazer os passos de inferência mais importantes e também porque é o aspeto com mais diferenças em relação à lógica clássica.

[O texto que se segue, a respeito da implicação vaga, dos métodos de composição (Max-Min, Max-θ e Max-λ) e dos métodos de inferência contém muito mais informação do que a considerada atualmente nas aulas de TSI. Infelizmente ainda não foi possível produzir um resumo com a informação estritamente necessária para as aulas e a avaliação]

Existem 15 implicações vagas distintas, com propriedades diferentes em relação à inferência. A tabela da Figura 7 apresenta as funções de verdade de (x is A)→(y is B) para algumas relações de implicação, em que A se define no universo de discurso U, e B se define no universo de discurso V. Na tabela foram usadas algumas abreviaturas: u, v em vez de μ(u), μ(v); ∧ em vez de Min e ∨, em vez de Max.

Ra	$1 \wedge (1 - u + v)$
Rm	$(u \wedge v) \vee (1 - u)$
Rc	$u \wedge v$
Rb	$(1 - u) \vee v$
Rs	1 se $u \leq v$; 0 se $u > v$
Rg	1 se $u \leq v$; v se $u > v$
Rgs	$[u \rightarrow v] \wedge [(1 - u) \rightarrow (1 - v)]$
	g S
Rgg	$[u \rightarrow v] \wedge [(1 - u) \rightarrow (1 - v)]$
	g G
Rsg	$[u \rightarrow v] \wedge [(1 - u) \rightarrow (1 - v)]$
	s G
Rss	$[u \rightarrow v] \wedge [(1 - u) \rightarrow (1 - v)]$
	s S

Figura 7 - Implicações vagas

Uma implicação vaga (x is A)→(y is B), em que A e B são dois conjuntos vagos definidos sobre os universos de discurso U e V, tem uma função de verdade com domínio U×V e contradomínio [0,1]. A função de verdade de (x is A)→(y is B) pode ser representada por uma matriz de duas entradas em que para cada par (u,v) se apresenta o grau de verdade de (x is A)→(y is B). A Figura 8 mostra as funções de pertença dos conjuntos vagos "heavy smoking" e "high risk of cancer", e a matriz que representa a função de verdade da implicação (smoking is “heavy smoking”) → (risk is “high risk of cancer”) para a relação de implicação Rc.

Heavy Smoking: {0/0, 2/0.1, 4/0.6, 6/0.8, 10/1}

High Risk of Cancer: {1/0, 2/0.2, 3/0.7, 4/0.9, 5/1}

	Risco					
Nº Cigarros		1	2	3	4	5
	0	0.0	0.0	0.0	0.0	0.0
	2	0.0	0.1	0.1	0.1	0.1
	4	0.0	0.2	0.6	0.6	0.6
	6	0.0	0.2	0.7	0.8	0.8
	10	0.0	0.2	0.7	0.9	1.0

Figura 8 - Grau de verdade da implicação RC "heavy smoker"→"high risk of cancer"

As implicações vagas nem sempre se comportam de acordo com a nossa intuição da lógica de primeira ordem. Por exemplo, algumas implicações não têm o Modus Ponens, e outras não têm o Modus Tollens. A tabela da Figura 9 apresenta as propriedades das várias implicações lógicas definidas na Figura 7, quando é usado uma determinada maneira de fazer inferência, designado por composição Max-Min (ver secção 4.2).

		Relação de implicação para A→B									
Premissa	Inferido	Ra	Rm	Rc	Rb	Rs	Rg	Rgs	Rgg	Rsg	Rss
A	B (MP)	-	-	+	-	+	+	+	+	+	+
very A	very B	-	-	-	-	+	-	-	-	+	+
very A	B	-	-	+	-	-	+	+	+	-	-
more or less A	more or less B	-	-	-	-	+	+	+	+	+	+
more or less A	B	-	-	+	-	-	-	-	-	-	-
not A	unknown	+	+	-	+	+	+	+	+	+	+
not A	not B	-	-	-	-	-	-	+	+	+	+
not B	not A (MT)	-	-	-	-	+	-	-	-	+	+
not very B	not very A	-	-	-	-	+	-	-	-	+	+
not more or less B	not more or less A	-	-	-	-	+	-	-	-	+	+
B	unknown	+	-	-	+	+	+	-	-	-	-
B	A	-	-	+	-	-	-	+	-	-	+

Figura 9 - Propriedades das implicações vagas

Pode constatar-se na Figura 9 que nem todas as implicações exibem o Modus Ponens (MP) clássico (quando é usado Max-Min). Apenas as implicações Rc, Rs, Rg, Rgs, Rgg, Rsg e Rss o têm. Destas, apenas a Rs, a Rsg e a Rss têm também o Modus Tollens (MT) clássico.

A implicação da lógica clássica exibe outras propriedades. Se tivermos a implicação $A \Rightarrow B$ e $\neg A$ não podemos concluir nada; se tivermos $A \Rightarrow B$ e B também não podemos concluir nada. As implicações vagas Rs, Rsg e Rgg têm simultaneamente o MP, o MT e estas duas propriedades da implicação clássica.

Algumas implicações vagas têm propriedades muito interessantes, úteis e que excedem as propriedades da lógica clássica. Para as implicações Rs, Rsg e Rgg, além das propriedades da lógica clássica já descritas, se tivermos $A \rightarrow B$ e *"very A"* podemos concluir *"very B"*; se tivermos $A \rightarrow B$ e *"more or less A"* podemos concluir *"more or less B"*; se tivermos $A \rightarrow B$ e *"not very B"* podemos concluir *"not very A"*; e se tivermos $A \rightarrow B$ e *"not more or less B"* podemos concluir *"not more or less A"*. *"very A"* e *"more or less A"* representam-se pelos modificadores de concentração e de diluição, respetivamente. Embora bastante intuitivas e desejáveis, a lógica clássica não tem estas propriedades, as quais permitem poupar muito o número de regras necessárias num sistema de lógica vaga.

Se, em vez do Max-Min, usarmos outro método de aplicação de regras de inferência (por exemplo Max-θ ou Max-λ, ver secção 4.2), as propriedades das implicações vagas podem ser outras.

A lógica vaga, tal como a lógica clássica, suporta a inferência de conclusões novas a partir de um conjunto existente de proposições vagas. No entanto, e ao contrário daquilo que se passa com a lógica clássica, os métodos de inferência usados na lógica vaga são métodos empíricos muito associados a sistemas concretos. Existem essencialmente dois métodos para a realização de inferência com proposições vagas, cada um deles com variações: a inferência decomposicional, ou inferência com valores vagos; e a inferência com valores exatos. Em qualquer deles, de um modo semelhante ao que se passou com as regras com fator de confiança, as conclusões que se podem produzir não dependem de um conjunto pequeno, local, de proposições, mas sim de todo o conjunto existente de proposições. Por essa razão, a inferência vaga é não monótona.

Na inferência decomposicional (i.e., inferência com valores vagos), os inputs do sistema de lógica vaga, os quais são em geral valores exatos (e.g., uma temperatura, uma taxa de juro, uma distância e uma velocidade), têm de ser convertidos em valores vagos. A este processo chama-se fuzzificação. Depois da

fuzzificação, determina-se a contribuição de cada implicação para o resultado final. Mas a determinação da contribuição de uma implicação para o resultado final usa um mecanismo que decompõe uma implicação com condições compostas (por exemplo, $(P \wedge Q \wedge R \vee S) \rightarrow T$) num conjunto de implicações com condições elementares (por exemplo, $P \rightarrow T$, $Q \rightarrow T$, $R \rightarrow T$, e $S \rightarrow T$), calcula a contribuição de cada uma destas implicações elementares para o resultado (usando regras de inferência como o Modus Ponens e o Modus Tollens) e combina as várias contribuições elementares para encontrar a contribuição da implicação original. É devido a este processo de decompor uma implicação complexa em diversas implicações mais simples que se designa este método por *inferência decomposicional*. Entretanto, a aplicação de regras de inferência usa sempre valores vagos, nunca os convertendo em valores exatos em nenhum passo do processo.

Depois de ter as contribuições de todas as regras, combinam-se essas contribuições numa conclusão única que será a conclusão final do sistema. O resultado deste processo é ainda um valor vago. Quando a contribuição final do sistema é computada, é necessário convertê-la num valor exato, pois os valores vagos não têm interesse prático na maioria das aplicações. À conversão de um valor vago num valor exato chama-se desfuzzificação.

Na inferência com valores exatos, os inputs exatos do sistema não são convertidos em valores vagos. Neste método de inferência, tal como no método decomposicional, cada implicação produz uma contribuição para o resultado final. No entanto as implicações com condições complexas não se decompõem em implicações mais simples. As implicações são usadas tal como foram originalmente escritas. Apesar de não se converterem os inputs exatos em valores vagos, na inferência com valores exatos, a contribuição de uma implicação é um valor vago.

Tal como no método decomposicional, depois de ter os valores vagos que resultam de todas as implicações, esses valores são combinados num único valor vago que representa o resultado final do sistema. Este valor vago é igualmente convertido num valor exato para se poder usar fora do sistema.

Como a inferência vaga está muito associada a sistemas concretos, explicações mais detalhadas dos métodos de inferência são adiados para depois da descrição dos sistemas de regras baseados em lógica vaga.

3 Tipos de regras em sistemas baseados em lógica vaga

[Atualmente, esta secção não é dada nas aulas de TSI]

Os sistemas baseados em lógica vaga são geralmente sistemas mistos que combinam os métodos de representação e de raciocínio exatos inspirados na lógica de primeira ordem com os métodos da lógica vaga. A base de conhecimentos de um sistema baseado em lógica vaga contem conhecimento representado através de regras exatas e de regras vagas com ou sem fator de confiança. O raciocínio de um sistema baseado em lógica vaga combina métodos de raciocínio exato (e.g., a dedução) e do raciocínio com fator de confiança com os métodos de raciocínio da lógica vaga.

Nesta secção são analisados vários tipos de regras vagas e métodos usados para efetuar raciocínio com regras vagas.

Vários tipos de regras baseadas na lógica vaga têm sido propostos por diversos autores. Entre estas, salientam-se as regras de Zadeh-Mamdani; as regras vagas com fatores de confiança; as regras de Takagi-Sugeno; as regras vagas graduais; as regras generalizadas; e as regras generalizadas com variáveis.

Regras de Zadeh-Mamdani

As regras de Zadeh-Mamdani têm o formato **`If`** `x is A` **`Then`** `y is B`, em que x e y são duas variáveis vagas definidas sobre os universos U e V respetivamente; e A e B são dois conjuntos vagos definidos pelas funções de pertenças $\mu_A: U \rightarrow [0,1]$, e $\mu_B: V \rightarrow [0,1]$.

Na sua forma mais geral, uma regra vaga de Zadeh-Mamdani tem o formato

If x_1 is A_1 AND x_2 is A_2 AND …AND x_n is A_n **Then** y is B

Regras vagas com fator de confiança

Uma regra vaga pode ser combinada com um fator de confiança expressando a confiança do sistema na regra. Uma regra de Zadeh-Mamdani com fator de confiança tem o formato

If x is A **Then** y is B (CF)

em que CF é o fator de confiança da regra.

As regras vagas com fator de confiança usam-se quando o conhecimento nelas representado não é absolutamente fiável (possivelmente porque é heurístico). Um exemplo de regra vaga com fator de confiança é a seguinte

If current economic situation is good, and
current political situation is good, and
the predicted value for tomorrow is up
Then action is buy (**CF** = 0.9).

Num sistema baseado em regras vagas com fator de confiança, os factos vagos também podem ter fator de confiança.

Regras de Takagi-Sugeno

O formato geral das regras de Takagi-Sugeno é o seguinte

If x_1 is A_1 AND … AND x_n is A_n **Then** y is $f(x_1,\ldots, x_n)$

Se a função *f* for linear, uma regra de Takagi-Sugeno passa a ter o seguinte formato:

If x_1 is A_1 AND … AND x_n is A_n **Then** $y=C_0+C_1.x_1+ \ldots+C_n.x_n$

As regras de Takagi-Sugeno são especialmente úteis para aproximação de funções.

Regras vagas graduais

As regras vagas graduais são regras de Zadeh-Mamdani, mas em vez de usarem valores vagos para as variáveis vagas, elas usam representações vagas de propriedades graduais, por exemplo: *"quanto mais vermelho é um tomate, mais maduro ele é"*; e *"quanto mais elevados são os fundos federais, mais elevada é a taxa de juro a curto prazo"*.

Regras condição-conclusão generalizadas

Por vezes, as proposições vagas individuais que constituem a condição de uma regra vaga não têm todas a mesma importância para a conclusão da regra. Quando isso acontece, é necessário indicar a importância de cada uma das proposições vagas da parte esquerda da regra. Coeficientes de importância relativa (DIi), fatores de tolerância ao ruido (NT – Noise Tolerance); coeficientes de sensibilidade (SF), para além de fatores de confiança têm sido usados em regras vagas:

If $(x_1$ is $A_1)DI_1$ AND …AND $(x_n$ is $A_n)DI_n$ **Then** A1, …, Ak (NT, SF, CF)

Regras condição-conclusão generalizadas com variáveis

As regras vagas também foram estendidas de modo a usarem variáveis em vez de constantes nos valores vagos das variáveis vagas.

Por exemplo, independentemente do valor do índice para ontem e para hoje, o seu valor será o mesmo para amanhã:

IF index_yesterday is V, and
index_today is V
THEN index_tomorrow is V (CF=0.8)

4 Raciocínio em sistemas baseados em lógica vaga

Numa base de conhecimentos de regras vagas, as regras ligam-se entre elas através de um "else-link", o qual não tem a mesma interpretação que o “else” de uma instrução “if-then-else” de uma linguagem de programação:

If $x_{1.1}$ is $A_{1.1}$ AND …AND $x_{1.n}$ is $A_{1.n}$ **Then** y_1 is B_1

If $x_{2.1}$ is $A_{2.1}$ AND …AND $x_{2.n}$ is $A_{2.n}$ **Then** y_2 is B_2

…

If $x_{m.1}$ is $A_{m.1}$ AND …AND $x_{m.n}$ is $A_{m.n}$ **Then** y_m is B_m

Durante o seu raciocínio, o sistema baseado em regras vagas usa todas as regras cujo lado direito contribuir para determinar a variável cujo valor se pretende conhecer. Todas essas regras contribuem para o valor do resultado final. Quando mais que uma regra contribui para o valor de uma variável vaga y, é necessário combinar todas as contribuições e produzir um resultado único. A forma de combinar os valores de uma variável, obtidos pela utilização de várias regras, depende do tipo de *"else-link"* usado. Existem diversos tipos de *"else-link"* muito utilizados: *"or-link"*, *"and-link"*, *"probor"*, *"truth qualifiation-link"*, e *"additive link"*, por exemplo. As propriedades do mecanismo de inferência vaga dependem do *"else-link"* usado. Um sistema baseado em lógica vaga deve possuir os vários tipos de *"else-link"* para que o engenheiro do conhecimento os possa ensaiar, e escolher o melhor entre eles para a sua aplicação.

A contribuição de uma regra para uma dada variável cujo valor se pretende conhecer é um conjunto vago. O resultado de combinar as contribuições das várias regras numa única contribuição, usando o *else-link*, é também um conjunto vago. No entanto, o sistema não deve dizer que o valor de uma dada variável é um conjunto vago pois, na maioria dos casos, isso seria muito pouco útil para o utilizador do sistema. É necessário primeiro converter um conjunto vago num valor exato. A esse processo chama-se desfuzzificação. Existem dois métodos de desfuzzificação muito conhecidos: o método dos máximos e o método do centro de massas (ou centroide).

Seguidamente apresentam-se os vários métodos propostos para o *else link* e os dois métodos propostos para a desfuzzificação. Posteriormente descrevem-se dois métodos de inferência usados para determinar a contribuição de cada regra para uma dada variável.

4.1 Else-links

Existem diversos métodos para implementar o *else-link*, entre os quais o "*or-link*" ou método dos máximos, o "*and-link*" ou método dos mínimos, o método da verdade qualificada (*truth-qualification link*), o método da soma ponderada, e o Probor (*probabilistic OR*, Probor(a, b) = a+b – a×b).

OR-link

O valor vago de uma variável vaga é o máximo dos resultados parciais obtidos pelas várias regras para essa variável vaga. Por exemplo, se devido a uma regra, o valor de uma variável é {1/0.2, 2/0.6, 3/0.8, 4/1, 5/1} e, se devido a uma segunda regra, o valor da mesma variável é {1/0.1, 2/0.7, 3/0.9, 4/0.9, 5/1}, o resultado das duas regras ligadas pelo "Or-link" é {1/0.2, 2/0.7, 3/0.9, 4/1, 5/1}.

AND-link

O valor vago de uma variável vaga é o mínimo dos resultados parciais obtidos pelas várias regras para essa variável vaga. Por exemplo, se devido a uma regra, o valor de uma variável é {1/0.2, 2/0.6, 3/0.8, 4/1, 5/1} e, se devido a uma segunda regra, o valor da mesma variável é {1/0.1, 2/0.7, 3/0.9, 4/0.9, 5/1}, o resultado das duas regras ligadas pelo "And-link" é {1/0.1, 2/0.6, 3/0.8, 4/0.9, 5/1}.

Método Probor

Para usar o método Probor, o grau de pertença de cada um dos membros do universo de discurso é o resultado de aplicar a função *Probor/2* aos graus de pertença correspondentes dos dois conjuntos que se pretende combinar. Exemplificamos seguidamente a aplicação do *Probor*, cuja definição é Probor(a, b) = a+b – a×b.

A = {0/0, 5/0, 10/0.24, 15/0.24}

B = {0/0.2, 5/0.2, 10/0, 15/0}

Probor: {0/0.2, 5/0.2, 10/0.24, 15/0.24}

Verdade qualificada ("Truth qualification link")

[Já não se dá em TSI]

A ideia principal deste tipo de *"else-link"* é a escolha da alternativa com o maior grau de verdade. Dado que o *"else-link"* determina a maneira como se combinam os vários valores de uma variável vaga num único valor, o universo de discurso de todos esses valores vagos alternativos é o mesmo. Assim sendo, cada regra i produz um resultado com importância Ti, o qual pode ser a soma dos graus de verdade do resultado para todos os elementos do universo de discurso. O valor final da variável é o valor determinado pela regra com o maior Ti.

Link aditivo

[Já não se dá em TSI]

A ideia subjacente a este *"else-link"* é efetuar uma média ponderada das várias contribuições obtidas pelas várias regras. As ponderações a usar para cada regra podem ser os Ti computados como no método da verdade qualificada.

Os valores Bi obtidos para a variável de saída são ponderados por fatores de ponderação: $\mu_{B'} = \Sigma\mu_{Bi'} \times w_i$. Os w_is poderão ser os T_is como determinados no *"link"* da verdade qualificada.

4.2 Inferência decomposicional

[O método de inferência decomposicional e o método de inferência com valores exatos partilham os últimos dois passos: else-link e desfuzificação. Esses dois últimos passos do método de inferência decomposicional podem, portanto, fazer parte de perguntas na avaliação. Os passos de fuzificação dos inputs, de decomposição, e de agregação também podem fazer parte de perguntas na avaliação. A determinação da contribuição de cada regra elementar não pode ser pedido na avaliação]

Resumo do método de inferência decomposicional

Suporemos um sistema com uma base de conhecimentos com as seguintes três regras vagas:

IF $x_{1.1}$ IS $A_{1.1}$ AND $x_{1.2}$ IS $A_{1.2}$ **THEN** y is B_1

IF $x_{2.1}$ IS $A_{2.1}$ OR $x_{2.2}$ IS $A_{2.2}$ **THEN** y is B_2

IF $x_{3.1}$ IS $A_{3.1}$ AND $x_{3.2}$ IS $A_{3.2}$ **THEN** z is C

O mecanismo de inferência pode ser encadeado para trás ou encadeado para a frente. Vamos optar por um mecanismo de inferência encadeado para trás cujo objetivo atual é a determinação do valor exato da variável vaga y.

Para isso, são escolhidas as regras que permitem contribuir para o valor da variável y, i.e., as regras 1 e 2. Cada uma delas será sujeita a um processo de inferência decomposicional, o que dá origem às quatro regras.

IF $x_{1.1}$ is $A_{1.1}$ **THEN** y is B_1

IF $x_{1.2}$ is $A_{1.2}$ **THEN** y is B_1

IF $x_{2.1}$ is $A_{2.1}$ **THEN** y is B_2

IF $x_{2.2}$ is $A_{2.2}$ **THEN** y is B_2

Usando um método de composição (e.g., o Max-Min) e escolhendo uma relação de implicação adequada, determina-se a contribuição de cada uma destas regras para a variável y, o que dá origem aos valores B_1', B_1'', B_2' e B_2''. Para que estes resultados parciais possam ser obtidos, há que saber qual a relação de implicação escolhida para que se possa determinar a função de verdade de cada uma das quatro regras.

B_1' e B_1'' são agregados através do *"and-link"* porque a condição da primeira regra original é uma conjunção. B_2' e B_2'' são agregados através do *"or-link"* porque a condição da segunda regra original é uma disjunção.

Finalmente, o resultado da agregação de B_1' e B_1'' e o resultado da agregação de B_2' e B_2'' são combinados através do *"else-link"* escolhido, dando origem ao valor vago de y.

De seguida, alguns dos passos da inferência decomposicional são apresentado em maior detalhe.

Decomposição e agregação

Usando o método de inferência decomposicional, se tivermos uma regra como

If x_1 is A_1 AND x_2 is A_2 AND …AND x_n is A_n **Then** y is B

devemos decompô-la nas regras

If x_1 is A_1 **Then** y is B

If x_2 is A_2 **Then** y is B

.....

If x_n is A_n **Then** y is B

Depois da decomposição, calculam-se as várias contribuições para y devidas a cada uma das n regras, e computa-se um resultado final para y, usando o *"and-link"*, dado que as condições do lado esquerdo da regra original estão conectadas através da conjunção. O resultado de cada regra é computado através de um método de composição (e.g., o max-min, explicado mais adiante).

De modo análogo, se tivermos a regra

If x_1 is A_1 OR x_2 is A_2 OR …OR x_n is A_n **Then** y is B

devemos decompô-la nas regras

If x_1 is A_1 **Then** y is B

If x_2 is A_2 **Then** y is B

.....

If x_n is A_n **Then** y is B

Depois, calculam-se as várias contribuições para y devidas a cada uma das n regras, e computa-se um resultado final para y, usando o *"or-link"*, dado que as condições do lado esquerdo da regra original estão conectadas através da disjunção.

O mesmo método pode ser usado se o lado esquerdo da regra tiver conjunções e disjunções, desde que o *"else-link"* usado seja correspondente à conectiva usada na condição da regra.

Ao processo de separação de uma regra com várias condições em várias regras chama-se *"decomposição"*. Ao processo de combinar os diversos resultados obtidos pelas várias regras da decomposição num único resultado de acordo com as conectivas da condição da regra original chama-se *"agregação"*.

Contribuição de cada regra elementar

[Não se dá]

A computação da contribuição de cada uma das regras elementares derivadas de uma regra mais complexa original faz-se através da aplicação de regras de inferência adaptadas ao raciocínio com lógica vaga. Existem diversas regras de inferência, mas as mais conhecidas são o Modus Ponens Generalizado e o Modus Tollens Generalizado.

Modus Ponens	Modus Tollens
$A \rightarrow B$ A' $B' = A' \circ (A \rightarrow B)$	$A \rightarrow B$ B' $A' = B' \circ (A \rightarrow B)$

Figura 10 - Modus Ponens e Modus Tollens Generalizados

Note-se que se (x is B) tem um dado grau de verdade não nulo, então not (x is B) também terá um grau de verdade possivelmente não nulo, daí a aparência pouco intuitiva do Modus Tollens generalizado.

O operador *"o"* da Figura 10 é o operador de composição. Existem diversos métodos de composição. Os mais usados no modus ponens generalizado são o Max-Min, o Max-θ, e o Max-λ, dos quais falaremos na secção seguinte.

No Modus Ponens da lógica clássica, se tivermos P⇒Q e P, podemos concluir Q. Mas o Modus Ponens da lógica vaga é muito mais flexível. Se tivermos P→Q e P', podemos concluir Q'. Qual é então a relação entre Q' por um lado, e P→Q e P' por outro lado?

Em termos gerais, diz-se que Q' é a composição de P' com P→Q, o que se indica da seguinte forma:

Q' = P' o P→Q

em que *"o"* representa a composição. Seguidamente descrevem-se três métodos para efetuar composição: Max-Min, Max-θ, e Max-λ.

Sendo P≡(x is A) e Q≡(y is B) em que A e B são definidos sobre os universos de discurso U e V, respetivamente, as funções de verdade de P e P' são aplicações de U para [0,1]; a função de verdade de P→Q é uma aplicação de U×V em [0,1]; e as funções de verdade de Q e de Q' são aplicações de V em [0,1]. Consequentemente, a composição combina μ_U:U→[0,1] com $\mu_{U\times V}$:U×V→[0,1] para obter μ_V:V→[0,1].

Max-Min

No método de composição Max-Min, a função de verdade de Q' obtém-se pela seguinte expressão:

$\mu_{Q'}(v)=MAX(\{Min(\mu_{P'}(u), \mu_{P\to Q}(u,v)): u\in U\})$

Para cada v, determina-se o conjunto dos mínimos entre $\mu_{P'}(u)$ e $\mu_{P\to Q}(u,v)$ para todos os elementos do universo de discurso U; o grau de verdade de Q' em v ($\mu_{Q'}(v)$) é o máximo do conjunto dos mínimos.

Vejamos qual o risco de cancro quando se fuma 4 cigarros, se tivermos a implicação Rc (smoking is heavy smoking)→(risk is high risk of cancer) e o método de composição Max-Min.

4 cigarros pode representar-se através de um conjunto vago cuja função de pertença é zero em todo o lado excepto quando o número de cigarros é 4, ponto em que toma o valor 1. **Smoking**: [0/0.0, 2/0.0, 4/1.0, 6/0.0, 10/0.0].

A função de verdade da implicação *(smoking is "high smoking")→(risk is "high risk of cancer")* representa-se na matriz apresentada na Figura 11.

	Risco					
Nº Cigarros		1	2	3	4	5
	0	0.0	0.0	0.0	0.0	0.0
	2	0.0	0.1	0.1	0.1	0.1
	4	0.0	0.2	0.6	0.6	0.6
	6	0.0	0.2	0.7	0.8	0.8
	10	0.0	0.2	0.7	0.9	1.0

Figura 11 - Grau de verdade da implicação "heavy smoker"→"high risk of cancer"

Vejamos qual é o grau de verdade de risco de cancro igual a 1 (v=1):

$\mu_{Q'}(1)=Max\{Min(\mu_{P'}(0), \mu_{P\to Q}(0,1)), Min(\mu_{P'}(2), \mu_{P\to Q}(2,1)), Min(\mu_{P'}(4), \mu_{P\to Q}(4,1)), Min(\mu_{P'}(6), \mu_{P\to Q}(6,1)), Min(\mu_{P'}(10), \mu_{P\to Q}(10,1))\}$

$\mu_{Q'}(1)=Max\{Min(0, 0), Min(0, 0), Min(1, 0), Min(0, 0), Min(0, 0)\}$

$\mu_{Q'}(1)=Max\{0, 0, 0, 0, 0\}$

$\mu_{Q'}(1)=0$

Seguidamente, determina-se o grau de verdade de risco de cancro igual a 2 (v=2).

$\mu_{Q'}(2)=Max\{Min(\mu_{P'}(0), \mu_{P\to Q}(0,2)), Min(\mu_{P'}(2), \mu_{P\to Q}(2,2)), Min(\mu_{P'}(4), \mu_{P\to Q}(4,2)), Min(\mu_{P'}(6), \mu_{P\to Q}(6,2)), Min(\mu_{P'}(10), \mu_{P\to Q}(10,2))\}$

$\mu_{Q'}(2)=Max\{Min(0, 0), Min(0, 0.1), Min(1, 0.2), Min(0, 0.2), Min(0, 0.2)\}$

$\mu_{Q'}(2)=Max\{0, 0, 0.2, 0, 0\}$

$\mu_{Q'}(2)=0.2$

O grau de verdade de risco de cancro igual a 3 é calculado da seguinte forma:

$\mu_{Q'}(3)=Max\{Min(\mu_{P'}(0), \mu_{P\rightarrow Q}(0,3)), Min(\mu_{P'}(2), \mu_{P\rightarrow Q}(2,3)), Min(\mu_{P'}(4), \mu_{P\rightarrow Q}(4,3)), Min(\mu_{P'}(6), \mu_{P\rightarrow Q}(6,3)), Min(\mu_{P'}(10), \mu_{P\rightarrow Q}(10,3))\}$

$\mu_{Q'}(3)=Max\{Min(0, 0), Min(0, 0.1), Min(1, 0.6), Min(0, 0.7), Min(0, 0.7)\}$

$\mu_{Q'}(3)=Max\{0, 0, 0.6, 0, 0\}$

$\mu_{Q'}(3)=0.6$

Para o grau de verdade de risco de cancro igual a 4 temos:

$\mu_{Q'}(4)=Max\{Min(\mu_{P'}(0), \mu_{P\rightarrow Q}(0,4)), Min(\mu_{P'}(2), \mu_{P\rightarrow Q}(2,4)), Min(\mu_{P'}(4), \mu_{P\rightarrow Q}(4,4)), Min(\mu_{P'}(6), \mu_{P\rightarrow Q}(6,4)), Min(\mu_{P'}(10), \mu_{P\rightarrow Q}(10,4))\}$

$\mu_{Q'}(4)=Max\{Min(0, 0), Min(0, 0.1), Min(1, 0.6), Min(0, 0.8), Min(0, 0.9)\}$

$\mu_{Q'}(4)=0.6$

Usando um método de cálculo análogo, o grau de verdade de risco de cancro igual a 5 é:

$\mu_{Q'}(5)=0.6$

Isto é, o risco de cancro para fumadores que fumam 4 cigarros por dia, de acordo com a regra *(smoking is "heavy smoking")→(risk is "high risk of cancer")* é o conjunto vago {1/0.0, 2/0.2, 3/0.6, 4/0.6, 5/0.6}, o que significa que o fumador de 4 cigarros tem um risco 1 em grau 0, tem um risco 2 em grau 0.2, tem um risco 3 em grau 0.6, tem um risco 4 em grau 0.6 e tem um risco 5 em grau 0.6.

É interessante repetir agora o processo para fumadores de 10 cigarros, para fumadores de 2 cigarros e para não fumadores (0 cigarros).

Para 10 cigarros, temos um risco {1/0.0, 2/0.2, 3/0.7, 4/0.9, 5/1.0}. Para 0 cigarros temos um risco {1/0.0, 2/0.0, 3/0.0, 4/0.0, 5/0.0}. E para 2 cigarros, o risco é {1/0.0, 2/0.1, 3/0.1, 4/0.1, 5/0.1}. Ou seja, o fumador de 10 cigarros tem um risco máximo com o máximo grau de verdade. O não fumador não tem qualquer risco de cancro (devido ao tabaco), enquanto que o fumador muito moderado (2 cigarros por dia) tem qualquer risco de cancro em baixo grau.

Neste exemplo torna-se claro o poder dos sistemas de regras vagas. Uma única regra (feita exclusivamente a pensar nos grandes fumadores) permite fazer previsões para uma enorme quantidade de situações.

Como veremos a respeito do *"else-link"* (secção 4.1), se tivermos um sistema com mais regras, as previsões seriam diferentes.

Apresentam-se de seguida, os métodos de composição Max-θ e Max-λ.

<u>Max-θ</u>

$\mu_{Q'}(v)=MAX\{Max(0, \mu_{P'}(u)+\mu_{P\rightarrow Q}(u,v)-1): u\in U\}$

<u>Max-λ</u>

Para definir este método de composição, é preferível definir, primeiro a seguinte função de pertença:

$\mu(u, v)=\mu_{P'}(u)$ se $\mu_{P\rightarrow Q}(u,v) = 1$

$\mu(u, v)=\mu_{P\rightarrow Q}(u,v)$ se $\mu_{P'}(u) = 1$

$\mu(u, v) = 0$ se $\mu_{P'}(u) < 1$ e $\mu_{P\rightarrow Q}(u,v)< 1$

Tendo esta definição, pode especificar-se o método de composição Max-λ:

$\mu_{Q'}(v)=MAX\{\mu(u, v): u\in U\}$

4.3 Inferência com valores exatos

Em geral, os inputs fornecidos a um sistema baseado em lógica vaga são *inputs* exatos (i.e., *crisp values*). No caso do exemplo do cálculo da gratificação, já referido no início no texto introdutório deste componente bibliográfico e na secção 1, os *inputs* do sistema serão os valores da qualidade da comida, numa escala de 0 a 3, e da qualidade do serviço, numa escala de 0 a 6.

Não pretendemos que o sistema nos diga "a gratificação deve ser generosa", ou "a gratificação deve ser média". Queremos que o sistema nos diga o valor exato da gratificação que devemos dar, por exemplo numa escala de 0 a 15% do valor da conta. Em geral, estamos interessados que o sistema, embora baseado em conjuntos vagos e em lógica vaga, nos dê um *output* exato.

É muito importante salientar que, se pretendemos saber o valor de uma dada variável de *output*, por exemplo, o valor da gratificação que devemos dar, todas as regras que contribuem para essa variável de *output* são usadas. As suas contribuições são combinadas para produzir um valor único de *output*. Isto significa que o resultado produzido pelo sistema será sempre dependente de todas as regras relevantes da sua base de conhecimentos. É possível que, se tivermos por exemplo 3 regras para a mesma variável de output, o resultado seja um; mas se tivermos mais regras para essa variável, o resultado seja outro. Isto é, os sistemas baseados em lógica vaga efetuam um tipo de raciocínio não monótono.

Daqui em diante descreve-se todo o processo de inferência com valores exatos desde a receção dos *inputs* até à produção dos outputs vagos e sua desfuzzificação. O processo é uma sequência de cinco passos: *i)* determinação do grau de verdade das proposições atómicas das condições das regras, *ii)* aplicação dos operadores lógicos da condição das regras vagas usadas, *iii)* determinação da contribuição de cada regra (valor vago), *iv)* combinação das contribuições de todas as regra num único output vago, e *v)* desfuzzificação do conjunto vago final (conversão do *output* vago num *output* exato).

Recorrendo ao exemplo da determinação do montante da gratificação num restaurante, vamos agora descrever cada um destes cinco passos envolvidos na inferência com valores exatos.

R1: If (food is GoodFood) AND (service is Excellent) Then (tip is Generous)

R2: If (food is PoorFood) OR (service is Poor) Then (tip is Cheap)

R3: If (service is Good) Then (tip is Average)

Figura 12 – Regras para determinar o valor da gratificação

A base de conhecimento de um sistema baseado na lógica vaga não consiste apenas de regras. Para cada variável envolvida nas regras, necessitamos do seu universo de discurso e das definições dos valores vagos que ela pode tomar.

Variável *food* (qualidade da comida)

Universo de discurso UFood = {0, 1, 2, 3}

GoodFood = {0/0, 1/0.2, 2/0.8, 3/1}

PoorFood = {0/1, 1/0.8, 2/0.2, 3/0}

Variável *service* (qualidade do serviço)

Universo de discurso UService = {0, 1, 2, 3, 4, 5, 6}

Excellent = {0/0, 1/0, 2/0, 3/0, 4/0.3, 5/1, 6/1}

Good = {0/0, 1/0.2, 2/0.8, 3/1, 4/0.8, 5/0.2, 6/0}

Poor = {0/1, 1/1, 2/0.3, 3/0, 4/0, 5/0, 6/0}

Variável *tip* (valor da gratificação em percentagem do custo da refeição)

Universo de discurso UTip = {0, 5, 10,15}

Generous = {0/0, 5/0, 10/0.3, 15/1}

Average = {0/0.1, 5/1, 10/1, 15/0.1}

Cheap = {0/1, 5/0.3, 10/0, 15/0}

Ao longo do exemplo, assumiremos que a qualidade da comida tem o valor exato 2 (numa escala de 0 a 3) e a qualidade do serviço tem o valor exato 4 (numa escala de 0 a 6).

Determinação dos valores de verdade das proposições atómicas

O valor de verdade de uma proposição atómica coincide com o grau de pertença do valor exato da variável vaga no conjunto vago que representa o conceito da proposição. Temos de conhecer as regras vagas relevantes e os respetivos conjuntos vagos relevantes para cada um dos *inputs*.

No sistema para determinar o valor da gratificação a dar a um empregado de mesa em função da qualidade da comida e da qualidade do serviço, temos dois *inputs* – a qualidade da comida e a qualidade do serviço – e as três regras que se voltam a apresentar por conveniência, na Figura 12.

Passamos de seguida à determinação dos valores de verdade das proposições atómicas das condições das regras. Para isso, vamos dar um nome simbólico a ada uma delas.

P1 ≡ (food is GoodFood)	$\theta_{P1}(2) = \mu_{GoodFood}(2) = 0.8$
P2 ≡ (service is Excellent)	$\theta_{P2}(4) = \mu_{Excellent}(4) = 0.3$
P3 ≡ (food is PoorFood)	$\theta_{P3}(2) = \mu_{PoorFood}(2) = 0.2$
P4 ≡ (service is Poor)	$\theta_{P4}(4) = \mu_{Poor}(4) = 0$
P5 ≡ (service is Good)	$\theta_{P5}(4) = \mu_{Good}(4) = 0.8$

Tabela 2 – Valores de verdade das proposições atómicas das condições das regras

A Tabela 2 mostra os valores de verdade das proposições atómicas envolvidas nas condições das regras, quando a qualidade da comida é 2 e a qualidade do serviço é 4.

Valor de verdade da condição das regras

A aplicação dos operadores lógicos da condição de uma regra tem o propósito de determinar o valor de verdade da condição para depois se poder determinar a contribuição da regra para a variável de saída. A condição da regra R1 é uma conjunção, a condição da regra R2 é uma disjunção, enquanto que a condição da regra R3 é uma simples proposição atómica. Assim, na regra R1, tem de se usar a definição do valor de verdade da conjunção para determinar o valor de verdade da sua condição. Na regra R2, usa-se a definição do valor de verdade da disjunção para determinar o valor de verdade da sua condição. Finalmente, o valor de verdade da condição da regra R3 é o valor de verdade da sua única proposição atómica.

Na secção 2, foram apresentados três métodos para calcular o valor de verdade de uma conjunção e 3 métodos para o valor de verdade da disjunção.

Regra	Condição	Método 1	Método 2	Método 3
R1 (food is GoodFood) AND (service is Excellent)	$\theta_{P1 \wedge P2}(2, 4)$	Mínimos $\text{Min}(\theta_{P1}(2), \theta_{P2}(4)) = \text{Min}(0.8, 0.3) = 0.3$	Produto $\theta_{P1}(2) \times \theta_{P2}(4) = 0.8 \times 0.3 = 0.24$	Produto limitado $\text{Max}(\theta_{P1}(2)+\theta_{P2}(4)-1, 0) = \text{Max}(0.8+0.3-1, 0) = 0.1$
R2 (food is PoorFood) OR (service is Poor)	$\theta_{P3 \vee P4}(2, 4)$	Máximos $\text{Max}(\theta_{P3}(2), \theta_{P4}(4)) = \text{Max}(0.2, 0) = 0.2$	Soma limitada $\text{Min}((\theta_{P3}(2)+\theta_{P4}(4), 1) = \text{Min}(0.2+0, 1) = 0.2$	Probor $\theta_{P3}(2)+\theta_{P4}(4) - \theta_{P3}(2) \times \theta_{P4}(4) = 0.2+0 - 0.2 \times 0 = 0.2$

Regra	Condição	Valor de verdade
R3 (service is Good)	$\theta_{P5}(4)$	0.8

Tabela 3 – Valor de verdade das condições das regras

Os valores de verdade dos antecedentes das regras serão usados para determinar a contribuição da regra para a variável de saída.

Contribuição de cada regra

Esta fase do processo corresponde grosso modo à aplicação do *Modus Ponens*: conhecendo valor de verdade da condição da regra, pretendemos determinar a sua contribuição. Usaremos dois métodos para determinar a contribuição de cada regra: o método dos mínimos (*truncate*, i.e., truncar o conjunto de saída) e o método do produto (*shrink*, i.e., escalar o conjunto de saída).

No método *truncate*, a contribuição da regra é representada pelo conjunto vago que se obtém truncando o conjunto vago referido na conclusão pelo valor de verdade da condição da regra. Isto é o mesmo que determinar o mínimo entre o valor de verdade da condição e cada um dos graus de pertença do conjunto.

No método *shrink*, a contribuição da regra é representada pelo conjunto vago que se obtém multiplicando o valor de verdade da condição pelos graus de pertença do conjunto envolvido na conclusão da regra. Através deste produto, obtém-se um conjunto vago que é uma réplica do conjunto original da conclusão da regra mas sujeito a um fator de escala igual ao valor de verdade da condição da regra.

Recordemos as três regras do nosso sistema:

R1: If (food is GoodFood) AND (service is Excellent) Then (tip is Generous)

R2: If (food is PoorFood) OR (service is Poor) Then (tip is Cheap)

R3: If (service is Good) Then (tip is Average)

Para cada uma destas regras, temos que conhecer o valor de verdade da condição e o conjunto vago que representa o valor vago da variável da conclusão da regra. A tabela seguinte sistematiza essa informação, para o caso particular em que se usa o método dos mínimos para determinar o valor de verdade da conjunção, e o método dos máximos para determinar o valor de verdade da disjunção.

Regra	Verdade da cond.	Conjunto vago da conclusão	Contribuição (truncar)	Contribuição (escalar)
R1	$\theta_{P1 \wedge P2}(2, 4) = 0.3$	Generous = {0/0, 5/0, 10/0.3, 15/1}	Tip = {0/0, 5/0, 10/0.3, 15/0.3}	Tip = {0/0, 5/0, 10/0.09, 15/0.3}
R2	$\theta_{P3 \vee P4}(2, 4) = 0.2$	Cheap = {0/1, 5/0.3, 10/0, 15/0}	Tip = {0/0.1, 5/0.2, 10/0, 15/0}	Tip = {0/0.2, 5/0.06, 10/0, 15/0}
R3	$\theta_{P5}(4) = 0.8$	Average = {0/0.1, 5/1, 10/1, 15/0.1}	Tip = {0/0.1, 5/0.8, 10/0.8, 15/0.1}	Tip = {0/0.08, 5/0.8, 10/0.8, 15/0.08}

Tabela 4 – Contribuições das regras para a determinação da gratificação

As últimas duas colunas da Tabela 4 (as duas mais à direita) representam as contribuições das três regras para a variável *Tip* (gratificação), usando o método *truncate* e o método *shrink*. No método *truncate*, a contribuição de uma regra obtém-se truncando o conjunto vago envolvido na conclusão da regra pelo valor de verdade da condição. Por exemplo, na primeira regra, trunca-se o conjunto que representa uma gratificação elevada (conjunto *Generous*) por 0.3, que é o valor de verdade da condição da regra. Sendo o valor vago da conclusão desta regra *Generous*, parte-se da sua definição

Generous = {0/0, 5/0, 10/0.3, 15/1}

No método *truncate*, os graus de pertença maiores que o valor de corte (0.3, neste caso) passam para o valor de corte; e os graus de pertença iguais ou inferiores ao valor de corte mantêm o seu valor. O grau de pertença do 0 no conjunto *Generous* é 0. Como 0 é menor que o valor de corte (0.3), o grau de pertença mantém-se. Os graus de pertença de 5 e de 10 são 0 e 0.3, mantêm-se porque são menores ou iguais ao valor de corte. O grau de pertença de 15 é 1. Como 1 é maior que o valor de corte, o grau de pertença do 15 passa a 0.3.

No método escalar, basta multiplicar os graus de pertença do conjunto vago referido na conclusão da regra pelo valor de verdade da condição da regra. $0 \times 0.3 = 0$, $0.3 \times 0.3 = 0.09$, $1 \times 0.3 = 0.3$. Consequentemente, a contribuição da primeira regra para a gratificação, pelo método escalar, é o conjunto {0/0, 5/0, 10/0.09, 15/0.3}.

Na Tabela 4, apresentam-se a contribuição das regras, no caso em que o valor de verdade da conjunção foi calculado pelo método dos mínimos e o da disjunção pelo método dos máximos. Mas muitas outras possibilidades podem ser consideradas. Ao todo, há nove possibilidades: para cada uma das três possibilidades da conjunção (mínimos, produto e produto limitado), há três possibilidades da disjunção (máximos, probor e soma limitada). Seriam necessárias mais oito tabelas como a Tabela 4 para cobrir todas as possibilidades.

Os outputs vagos da variável *gratificação* (*tip*) – um por cada uma das três regras existentes para essa variável – serão combinados de modo que o sistema possa ter um único output para cada variável. Essa combinação designa-se por *Else-Link*.

Else-Link: Combinação dos Outputs Vagos Parciais

Num sistema baseado em conhecimento com regras vagas, existirão tantas contribuições para cada variável de saída quantas as regras para essa variável. No entanto, espera-se que o sistema produza uma resposta única para cada variável. Para isso, é necessário combinar os outputs vagos de todas as regras vagas num único output vago. O processo de combinar os outputs vagos de todas as regras que determinam o valor de uma dada variável chama-se *else-link* (ver secção 4.1).

A tabela mostra os resultados da combinação dos outputs das regras R1 a R3, usando o *and-link*, *or-link* e o *Probor*. Nesta tabela é usado sempre o resultado que se obtém de cada regra se for considerado o método truncar para determinar a contribuição de cada regra. Por outro lado, apenas serão considerados os valores de verdade da condição, obtidos pelo método dos mínimos para a conjunção e pelo método dos máximos para a disjunção.

Contribuição	And-link	Or-link	Probor-link
Tip = {0/0, 5/0, 10/0.3, 15/0.3}	Tip = {0/0, 5/0, 10/0, 15/0}	Tip = {0/0.1, 5/0.8, 10/0.8, 15/0.8}	Tip = {0/0.19, 5/0.84, 10/0.86, 15/0.37}
Tip = {0/0.1, 5/0.2, 10/0, 15/0}			
Tip = {0/0.1, 5/0.8, 10/0.8, 15/0.1}			

Tabela 5 – Combinação dos resultados de três regras

A aplicação do *and-link* consiste em, para cada elemento do universo de discurso, escolher o menor grau com que esse elemento pertence aos conjuntos que representam as contribuições das regras. Por exemplo, o 0 pertence em grau 0 à contribuição da primeira regra, pertence em grau 0.1 à contribuição da segunda regra, e pertence em grau 0.8 à contribuição da terceira regra. No resultado final, escolhe-se o menor destes graus de pertença, isto é, 0.

No *or-link*, em vez de se escolher o menor grau de pertença, escolhe-se o maior.

Para o *probor-link*, usa-se um processo de composição em que as contribuições das duas primeiras regras são operadas com o probor-link, produzindo um resultado que é operado com a contribuição da terceira regra para chegar ao resultado final. A aplicação sucessiva do *probor-link* é independente da ordem pela qual as várias contribuições são consideradas. Em vez de se ter combinado as duas primeiras contribuições e depois ter combinado o resultado com a terceira, poderíamos ter combinado por exemplo, as duas últimas contribuições e depois ter combinado o resultado com a primeira. O resultado seria o mesmo. De seguida apresenta-se a combinação das duas primeiras contribuições, aplicando o *probor-link*.

{0/0, 5/0, 10/0.3, 15/0.3}

{0/0.1, 5/0.2, 10/0, 15/0} (probor-link)

{0/0.1, 5/0.2, 10/0.3, 15/0.3}

Neste processo, o grau de pertença de cada elemento do universo de discurso é o Probor dos graus de pertença das duas contribuições. Recordemos, que Probor(a, b) = a+b – a×b. Por exemplo, o grau de pertença do 10 é 0.3 = 0.3+0 – 0.3×0.

Agora combina-se o resultado da combinação das duas primeiras contribuições com a terceira contribuição:

{0/0.1, 5/0.2, 10/0.3, 15/0.3}

{0/0.1, 5/0.8, 10/0.8, 15/0.1} (probor-link)

{0/0.19, 5/0.84,10/0.86, 15/0.37}

O último passo do processo de inferência consiste da desfuzzificação do conjunto vago que resulta da combinação das contribuições de todas as regras num único conjunto vago. Este processo de desfuzzificação converte um conjunto vago num valor exato.

Desfuzzificação

Em geral, esperamos que um sistema, quer seja baseado em lógica vaga quer não seja, nos dê um valor exato como output. Se pretendermos saber quanto devemos aquecer um quarto, pretendemos o valor exato de uma temperatura. Se pretendermos saber que força devemos aplicar num travão para travar um veículo, temos de ter um valor exato da força. Se pretendemos saber que gratificação dar ao empregado do restaurante, pretendemos também um valor exato. Para tal, dado que o resultado do processo de aplicação das regras vagas aos inputs produz um conjunto vago (quer se use o método de inferência decomposicional, quer se use a inferência com inputs exatos), é necessário converter o valor vago de saída num valor exato a dar ao utilizador. A este processo chama-se desfuzzificação. Consideraremos dois métodos de desfuzzificação – o método do centro de massas (ou centroide), e o método dos máximos.

O método do centroide consiste em determinar o centro de massas do conjunto vago que se pretende desfuzzificar. O método dos máximos consiste em determinar a média dos valores do universo de discurso com o valor de pertença mais elevado no conjunto vago que se pretende desfuzzificar.

Para melhor explicar os dois métodos, vamos desfuzzificar alguns dos conjuntos obtidos para a variável *Tip* (gratificação) por métodos alternativos.

Consideremos primeiro o resultado do *else-link*, quando se usa o *and-link* (Tabela 5): Tip = {0/0, 5/0, 10/0, 15/0}. É impossível aplicar o método do centroide porque envolve uma indeterminação:

$(0\times0 + 5\times0 + 10\times0 + 15\times0) / (0 + 0 + 0 + 0) = 0/0$

Poderemos pensar numa maneira de levantar esta indeterminação através do cálculo de limites. Em vez da aplicação *tout court* do método do centroide, podemos determinar o limite, quando o grau de pertença g tende para zero da expressão $(0\times g + 5\times g + 10\times g + 15\times g) / (g + g + g + g)$:

$\lim_{g \to 0} (0\times g + 5\times g + 10\times g + 15\times g) / (g + g + g + g) = 7.5$

Curiosamente, aplicando o método dos máximos, obtemos exatamente o mesmo valor: Seja M, o conjunto dos elementos do universo de discurso da gratificação com maior grau de pertença no conjunto vago que representa a gratificação, M = {0, 5, 10, 15} (todos os elementos têm o mesmo grau de pertença). O valor exato da gratificação é a média aritmética dos elementos de M: (0+5+10+15)/4 = 7.5.

Apesar da coincidência dos valores produzidos pelos dois métodos (7.5% do valor da refeição), podemos interrogar-nos se será que é lícito que o sistema dê uma resposta, nestas circunstâncias? O que significa a crença do sistema de que o valor da gratificação deve ser {0/0, 5/0, 10/0, 15/0}? Significa que o sistema acha que 0 pertence em grau 0 à gratificação que deve ser dada; 5% pertence em grau 0 à gratificação que deve ser dada; 10% pertence em grau zero à gratificação que deve ser dada; e 15% também não deve ser dado. Ou seja, não há nenhuma gratificação que o sistema ache que deva ser dada.

Além disso, 7.5% do valor da refeição não é uma resposta que gostássemos de obter porque, os *inputs* (qualidade da comida = 2 e qualidade do serviço = 4) configuram uma situação claramente melhor que a média. O valor mais representante da gratificação média é 10% do custo da refeição, o que nos levaria a esperar uma gratificação melhor que 10% mas não muito.

Tendo em conta, os argumentos apresentados, torna-se difícil aceitar que o sistema apresente uma sugestão nestas circunstâncias. No entanto, o conhecimento do sistema é bastante desajeitado – tem muito poucas regras, as regras cobrem muito poucas situações diferentes, e há uma grande sobreposição entre a primeira e a terceira regra. Daí que a má resposta possa ser mais o resultado das deficiências do conhecimento do que o resultado de uma má configuração.

Em vez de usarmos o valor vago da gratificação que resultou do *and-link*, podemos usar qualquer um dos outros. Vamos agora ver o que se passa com o resultado do *probor-link*, caso em que o valor vago da gratificação é Tip = {0/0.19, 5/0.84, 10/0.86, 15/0.36}.

Pelo método do centroide, o valor exato da gratificação é a soma ponderada

$$\text{Tip} = (0\times0.19 + 5\times0.84 + 10\times0.86 + 15\times0.37) / (0.19 + 0.64 + 0.86 + 0.37) = 8.91$$

Este valor é melhor do que o anterior, mas não é o que esperaríamos obter, o que reforça a suposição de que o conhecimento é muito deficiente.

Se, em vez do método do centroide, escolhermos o método dos máximos, a resposta ainda é melhor, mas esperaríamos obter uma gratificação ligeiramente maior. Seja M, o conjunto dos elementos do universo de discurso da gratificação com maior grau de pertença no conjunto vago que representa a gratificação, M = {10}. O valor exato da gratificação é a média aritmética dos elementos de M, ou seja 10.

Se tivéssemos usado, a gratificação vaga que resulta do *or-link* (i.e., {0/0.1, 5/0.8, 10/0.8, 15/0.8}), o valor exato da gratificação seria 9.6 = (0×0.1 + 5×0.8 + 10×0.8 + 15×0.8)/(0.1+0.8+0.8+0.8), usando o método do centroide, e 10 = Media({5, 10, 15}), usando o método dos máximos.

Estes exemplos, ainda que muito poucos (pois há muitíssimas mais configurações), parecem significar que o conhecimento representado não é o melhor.

Afinação do sistema

As regras usadas preveem poucas condições e há uma grande dose de sobreposição entre duas dela. Estas duas circunstâncias não são favoráveis. Além disso, os universos de discurso e os conjuntos vagos das variáveis *food* e *service* também não parecem os melhores. O facto de os universos terem um número diferente de elementos não parece ter justificação. Mais ainda, usar universos com um número par de elementos dificulta a definição de três valores vagos para as variáveis. A proposta de representação que se segue pretende colmatar estas deficiências. Considera-se o novo conjunto de regras que se apresenta de seguida.

IF (food IS GoodFood AND service IS Excellent) THEN (tip IS Generous)

IF (food IS AverageFood AND service IS Good) THEN (tip IS Average)

IF (food IS AverageFood AND service IS Poor) THEN (tip IS Cheap)

IF (food IS PoorFood AND service IS Good) THEN (tip IS Cheap)

IF (food IS PoorFood AND service IS Poor) THEN (tip IS None)

para as quais se torna necessário definir os conjuntos vagos que representam os valores imprecisos AverageFood e None. Além da alteração das regras, propomos também alterar as definições dos universos de discurso e dos conjuntos vagos associados às variáveis. Os universos das variáveis *food* (qualidade da comida) e *service* (qualidade do serviço) e correspondentes conjuntos vagos passaram a ser idênticos, pois a sua natureza é semelhante. Optamos por universos de 5 elementos, o que facilita a definição de três valores imprecisos. As novas definições das variáveis passaram a ser as que se seguem.

Variável *food*

Universo UFood = {1, 2, 3, 4, 5}

GoodFood = {1/0, 2/0, 3/0.3, 4/0.8, 5/1}
AverageFood = {1/0, 2/0.5, 3/1, 4/0.5, 5/0}
PoorFood = {1/1, 2/0.8, 3/0.2, 4/0, 5/0}

Variável *service*

Universo UService = {1, 2, 3, 4, 5}

Excellent = {1/0, 2/0, 3/0.3, 4/0.8, 5/1}
Good = {1/0, 2/0.5, 3/1, 4/0.5, 5/0}
Poor = {1/1, 2/0.8, 3/0.2, 4/0, 5/0}

Variável *tip*

Universo UTip = {0, 5, 10,15}

Generous = {0/0, 5/0, 10/0.3, 15/1}
Average = {0/0, 5/0.3, 10/1, 15/0.2}
Cheap = {0/0.2, 5/1, 10/0.2, 15/0}
None = {0/1, 5/0.2, 10/0, 15/0}

Com a nova definição dos universos de discurso, a situação que temos estado a testar (i.e., uma situação um pouco melhor do que média) será melhor representada por uma qualidade da comida com valor 3 e por uma qualidade do serviço com valor 5 (ou 4, por exemplo). Com esta nova base de conhecimento, usando o método dos mínimos para a conjunção e o método dos máximos para a disjunção, o método truncar para a contribuição das regras, o *probor-link* para o *else-link*, e o método do centroide para a desfuzzificação, obtém-se uma gratificação de 12,5% quando a qualidade da comida é 3 e a qualidade do serviço é 5. Este valor, de todos os determinados até agora, é sem dúvida o melhor e corresponde à nossa intuição, já que 12,5% é uma gratificação um pouco melhor que a média (10%). Se usarmos dois inputs iguais a 4, a gratificação é de 11,5% o que é também aceitável.

Se, em vez do *probor-link*, usarmos o *or-link*, obtemos resultados muito semelhantes para estes dois conjuntos de valores de input (12.5% e 11,6%, respetivamente). Infelizmente, quando se usa o *and-link*, o sistema não dá nenhum resultado. Com o novo conjunto de regras, há muito menor sobreposição entre as suas zonas de atuação. Ou seja, para qualquer conjunto de valores de entrada, há sempre regras cuja contribuição é 0. Consequentemente, o *and-link* origina conjuntos vagos com todos os graus de pertença iguais a 0, o que inviabiliza, como já vimos, a desfuzzificação pelo método do centroide.

5 Características dos sistemas baseados em lógica vaga

Nas secções 1 a 4 apresentamos os conjuntos vagos, a lógica vaga e os sistemas baseados em conhecimento que usam regras vagas para representar e raciocinar com conhecimento envolvendo imprecisão.

Esta secção apresenta alguns comentários finais. A primeira conclusão é que a abordagem baseada nos conjuntos vagos e na lógica vaga nos permite de facto fazer sistemas que são capazes de representar e processar conhecimento com conceitos imprecisos tal como acontece com as pessoas.

Além disso, apesar de termos escrito regras apenas para um determinado conjunto de situações, o sistema funcionou bem em situações não explicitamente previstas nas regras. Por exemplo, para uma comida com qualidade média (3) e um serviço com qualidade elevada (5), o sistema produziu uma gratificação de 12.5% do custo da refeição, o que é perfeitamente intuitivo. No entanto, não há regras para esta situação.

Mas há mais a dizer a este respeito.

Os sistemas baseados na lógica vaga efetuam, como já se disse, raciocínio não monótono, pois os seus resultados são dependentes da totalidade das suas bases de conhecimentos e não apenas de um conjunto bem definido e geralmente pequeno de premissas. Se inserirmos, na base de conhecimentos, novas regras para uma determinada variável de saída, é bem natural que os resultados produzidos pelo sistema para essa variável sejam diferentes dos que seriam obtidos antes da introdução dessas novas regras.

Tal como acontece com os sistemas de regras com fatores de confiança, os sistemas baseados na lógica vaga integram-se fundamentalmente no paradigma da representação simbólica, mas possuem igualmente um componente não simbólico. O resultado de cálculos com graus de pertença são realizados através de métodos empíricos que não derivam das propriedades formais do sistema de representação e, por isso, os valores dos graus de pertença dos conjuntos vagos produzidos como output não têm qualquer relação de representação com o mundo. Ainda que se possa argumentar que uma função de pertença de um valor vago de uma variável representa o grau em que cada elemento do universo de discurso pertence a esse conceito, dado o carácter empírico dos métodos usados para produzir os conjuntos vagos de saída, é muito difícil senão impossível manter a posição de que as funções de pertença dos conjuntos vagos de saída representem a realidade. Mais ainda, em geral, os resultados produzidos pelo sistema nem sequer são os conjuntos vagos que resultam da inferência; os resultados obtêm-se da desfuzzificação de conjuntos vagos, usando métodos igualmente empíricos.

Tendo um componente não simbólico associado a métodos empíricos que não resultam das propriedades formais do sistema de representação, os sistemas baseados em lógica vaga têm de ser afinados antes de poderem ser usados pelos seus utilizadores finais. Dada a enorme quantidade de escolhas que fazemos (e.g., inferência decomposicional vs. inferência com valores exatos; funções de verdade da conjunção e da disjunção, método usado para obter a contribuição de uma regra; método usado para combinar vários resultados vagos num único resultado; e método de desfuzzificação), é muito difícil afinar um sistema baseado em lógica vaga. Além de todas estas escolhas relativas ao seu funcionamento, a qualidade dos resultados produzidos depende ainda da definição dos conjuntos vagos usados para representar conceitos imprecisos, e das próprias regras do sistema. Se um resultado não é aquele que sabemos que deveria ser, quais destes fatores são responsáveis pela discordância?

Os sistemas baseados em lógica vaga têm a propriedade da degradação suave do comportamento ("*graceful degradation*"). Esta propriedade é uma vantagem, no sentido em que o sistema é capaz de dar respostas mesmo para situações para as quais não foi desenhado. Isso aumenta certamente a robustez do sistema mas, ao mesmo tempo, pode constituir uma armadilha para o utilizador. É que, estando sempre preparado para dar respostas, pode muito bem acontecer que a qualidade da resposta em situações muito afastadas daquelas para que foi concebido não tenha a qualidade suficiente. Isto pode levar o utilizador a fazer erros devido a sugestões dadas pelo sistema, as quais poderão ter sido produzidas sem conhecimento suficiente para dar respostas.

Por que razão o comportamento dos sistemas baseados na lógica vaga se degrada suavemente, tornando-os mais robustos que os sistema baseados em lógica de predicados de primeira ordem? Há muitos fatores que contribuem para que os sistemas baseados em lógica vaga tenham tendência para terem sempre resposta para as perguntas que lhes fazem, mesmo que não tenham sido desenhados a pensar nos problemas que têm de enfrentar. Consideremos de novo as regras e os valores vagos usados na primeira versão do sistema para determinar o valor da gratificação a dar num restaurante, os quais se reproduzem na Figura 13.

R1: If (food is GoodFood) AND (service is Excellent) Then (tip is Generous)

R2: If (food is PoorFood) OR (service is Poor) Then (tip is Cheap)

R3: If (service is Good) Then (tip is Average)

Variável *food* (qualidade da comida)

Universo de discurso UFood = {0, 1, 2, 3}

GoodFood = {0/0, 1/0.2, 2/0.8, 3/1}

PoorFood = {0/1, 1/0.8, 2/0.2, 3/0}

Figura 13 – Regras para determinar o valor da gratificação

Podemos ver, por exemplo que não há regras para casos em que a comida tem uma qualidade intermédia. As regras existentes apenas preveem situações em que a comida tem boa qualidade ou em que a comida tem má qualidade. Podemos dizer que o sistema não foi desenhado a pensar em situações com qualidade intermédia.

Apesar de não ter sido desenhado a pensar em comida com qualidade intermédia, se introduzirmos uma qualidade intermédia de *input* (digamos 1.5), o sistema dará resposta porque poderá executar todos os cinco passos que constituem o seu processamento. Se a comida tiver uma qualidade 1.5, o valor de verdade das condições (food is GoodFood) e (food is PoorFood) pode ser determinado. Usando os conjuntos GoodFood e PoorFood das secções 1 e 4.3, GoodFood = {0/0, 1/0.2, 2/0.8, 3/1} e PoorFood = {0/1, 1/0.8, 2/0.2, 3/0}, podemos assumir que a qualidade 1.5 teria um valor de pertença entre 0.2 e 0.8, digamos 0.5 no conjunto GoodFood, e entre 0.8 e 0.2, digamos 0.5 no conjunto PoorFood. Assim, as proposições (food is GoodFood) e (food is PoorFood) teriam ambas o valor de verdade 0.5.Tendo obtido o valor de verdade destas proposições, seria possível determinar os valores de verdade das condições das regras R1 e R2. Por outro lado, o valor de verdade da condição de R3 não depende da qualidade da comida, por isso poderia ser igualmente determinado se tivéssemos um valor para a qualidade do serviço. Conclui-se que, para uma situação intermédia, em termos da qualidade de comida, situação para a qual as regras do sistema não terão sido explicitamente desenhadas, o sistema poderia mesmo assim dar uma resposta.

Tendo os valores de verdade das condições das regras R1, R2, e R3, seríamos capazes de produzir os conjuntos vagos que se obtêm de cada uma dessas regras. Tendo as três contribuições, seríamos capazes

de obter o conjunto vago que resulta da sua combinação através do *else-link* escolhido. E, depois, seríamos capazes de desfuzzificar esse conjunto e obter o resultado. Conclui-se pois que o sistema é capaz de dar resposta mesmo em situações para que não foi pensado.

Outra forma de apresentar o que acabamos de dizer consiste em notar que, em geral, um sistema baseado em lógica vaga necessita de muito menos regras do que um sistema baseado na lógica clássica. Como acabamos de ver, um sistema baseado em lógica vaga dá respostas mesmo em situações não explicitamente contempladas pelas regras da base de conhecimento. Mas mais ainda, implícitos num conjunto vago usado numa regra, estão contempladas uma variedade de situações relativas a todos os elementos do seu universo de discurso. Para obter o mesmo efeito com regras baseadas em lógica clássica teríamos de escrever regras a pensar nos elementos todos do universo de discurso, o que resultaria certamente numa muito maior quantidade de regras.

Em conclusão, os sistemas baseados em lógica vaga são difíceis de afinar mas, depois de convenientemente afinados, são muito robustos, sendo mesmo capazes de se comportar bem em situações para as quais não foram explicitamente desenhados. Por outro lado, é necessário escrever muito menos regras para um sistema baseado em lógica vaga do que para um sistema equivalente baseado na lógica clássica, o que dá muito menos trabalho na fase da escrita das regras e obriga ao processamento de menos regras. No entanto, o processamento de inferência num sistema com um conjunto de regras vagas é mais do que o processamento que seria envolvido num sistema baseado em lógica clássica que possuísse o mesmo número de regras.

Dada a possibilidade de dar respostas mesmo em situações para que não terão sido explicitamente pensados, a utilização de sistemas baseados em lógica vaga é simultaneamente mais robusta e possivelmente mais arriscada.

6 FUZ-is: Inferência com valores exatos

O *FUZ-is* (Fuzzy Inference System) é um sistema de raciocínio e representação de conhecimento através de regras baseadas em lógica vaga.

O *FUZ-is* usa uma representação de regras condição-conclusão encadeadas para trás e um método de inferência com valores exatos. Cada variável usada numa regra será definida através da especificação do seu universo de discurso, dos seus valores vagos e, no caso de variáveis de saída, do método de desfuzzificação.

A condição de uma regra pode envolver apenas literais (e.g., not(x is A)), ou combinações arbitrárias de conjunções, disjunções e negações. A conclusão de uma regra pode ser um único literal.

Os vários passos da inferência do *FUZ-is* podem ser realizados através de métodos a especificar ao sistema, na sua configuração. O *FUZ-is* dispõe de vários métodos para a conjunção, a disjunção, a determinação da contribuição das regras com base nos seus antecedentes, o Else-Link, e a desfuzzificação. O programador poderá especificar métodos alternativos a estes, tendo para isso, que implementar os predicados correspondentes.

6.1 Definição de variáveis vagas

As variáveis vagas podem ser definidas através de duas alternativas: definição não estruturada, usando os predicados *variableUniverse/2*, *variableValueDef/3*, e *variableDefuzMethod/2;* ou a definição estruturada, usando o predicado *fuzzyVarDef/2* em combinação com os procedimentos *loadVariableDefinitions/0* e *clearVariableDefinitions/0*.

A definição não estruturada consiste de um conjunto de factos dos predicados *variableUniverse/2*, *variableValueDef/3*, e *variableDefuzMethod/2*, cada um dos quais associa a variável com uma parte da sua definição. A definição estruturada de uma variável usa o predicado fuzzyVardef/2 que associa uma variável com todos os componentes da sua definição.

A experiência de ensino com turmas de anos anteriores aconselha que os alunos a usem a primeira alternativa, i.e., a definição não estruturada, recorrendo aos predicados *variableUniverse/2*, *variableValueDef/3*, e *variableDefuzMethod/2*. Apesar de a definição estruturada poder parecer mais arrumada, tem-se revelado que a sua utilização é mais sujeita a erros. Com efeito, a definição estruturada obriga à execução judiciosa dos procedimentos *loadVariableDefinitions/0* e *clearVariableDefinitions/0*,

de cada vez que se alteram as definições. Basta que, por esquecimento, estes procedimentos não sejam convenientemente executados para que o sistema passe a funcionar de forma diferente da espetável após modificações efetuadas.

Definição não estruturada de variáveis vagas

Nesta abordagem, a definição de uma variável consiste de um conjunto de factos:

Um facto do predicado *variableUniverse/2* que define o universo de valores em que variável e os seus valores vagos são definidos;

Diversos factos do predicado *variableValueDef/3*, cada um dos quais define o conjunto vago que representa um dos valores vagos que variável pode usar; e

Se a variável for uma variável de saída, cujo valor tem de ser desfuzzificado, um facto do predicado *variableDefuzMethd/2* que associa a variável a um método de desfuzzificação.

De seguida apresentam-se exemplos da definição da variável que representa o tempo de evaporação de água no chão em condições ambientais, *evaporatio_time*.

Sendo um tempo de evaporação à temperatura ambiente, de pequenas quantidades de água, no chão, consideramos que o tempo poderá ter os valores {15, 30, 45, 60, 75, 90, 105}, em minutos. Este universo de discurso define-se através do seguinte facto:

```
variableUniverse(evaporation_time, [15, 30, 45, 60, 75, 90, 105]).
```

Assumiremos que esta variável poderá ter os valores vagos very_short, short, medium, long, e very_long. Para isso, teremos cinco factos de *variableValueDef/3*:

```
variableValueDef(evaporation_time, very_short,
   [15/1, 30/0.4, 45/0, 60/0, 75/0, 90/0, 105/0]).

variableValueDef(evaporation_time, short,
   [15/0.3, 30/1, 45/0.3, 60/0, 75/0, 90/0, 105/0]).

variableValueDef(evaporation_time, medium,
   [15/0, 30/0, 45/0.3, 60/1, 75/0.3, 90/0, 105/0]).

variableValueDef(evaporation_time, long,
   [15/0, 30/0, 45/0, 60/0, 75/0.3, 90/1, 105/0.3]).

variableValueDef(evaporation_time, long,
   [15/0, 30/0, 45/0, 60/0, 75/0, 90/0.4, 105/1]).
```

Assumindo que se trata de um sistema que determina o tempo de evaporação da água que molha o chão, dependendo da temperatura ambiente e da humidade relativa, a variável *evaporation_time* é a variável de saída. Consequentemente, requere a especificação de um método de desfuzzificação:

```
variableDefuzMethod(evaporation_time, peak).
```

Neste caso, foi especificado o método dos máximos (peak). E for preferido o método do centroide, recorre-se ao átomo *centroid* como argumento de *variableDefuzMethod/2*.

As outras variáveis teriam de ser definidas de maneira análoga, com a possível exceção que poderiam não necessitar do método de desfuzzificação.

A próxima secção descreve a especificação estruturada de variáveis vagas.

Definição estruturada de variáveis vagas

[Este não é o método sugerido para a definição de variáveis vagas; não poderão haver perguntas sobre ele]

fuzzyVarDef/2 associa uma varável vaga à sua definição. fuzzyVarDef(VarName, VarDef) significa que a variável vaga cujo nome é VarName tem a definição VarDef.

VarName é um átomo. VarDef é uma lista de pares (atributo : valor). Existem os seguintes atributos possíveis: *universe*, o qual especifica o universo de discurso da variável; *quantization*, que especifica os conjuntos vagos que representam os valores vagos da variável; e *defuz*, que especifica o método de desfuzzificação da variável, se existir.

O atributo ***universe*** serve para especificar o universo de discurso da variável e consequentemente de todos os seus valores vagos. O universo de discurso é especificado através de uma lista de valores, por exemplo [0, 10, 20, 30, 40].

O atributo ***quantization*** serve para especificar os conjuntos vagos que representam os valores vagos da variável. Uma quantização é uma lista de pares (setName : setDef) em que setName é o nome do valor vago definido e setDef é uma lista de pares valor / pertença. Nestes pares, *valor* é um elemento do universo de discurso e *pertença* é o grau de pertença desse valor no conjunto definido. Por exemplo, a quantização da variável *temperature* poderia ter os valores *low*, *medium* e *high*, em que por exemplo, o valor *low* seria definido como [0/1, 10/0.7, 20/0.2, 30/0, 40/0].

O atributo ***defuz***, se existir, especifica o método de desfuzzificação para variáveis de saída. Existe o método do centro de massas (*centroid*) e o método dos máximos (*peak*). Se o método de desfuzzificação não for especificado, o sistema, se necessário, usa o método do centro de massas (*centroid*).

Exemplos

```
fuzzyVarDef(temperature, [
  universe : [0, 10, 20, 30, 40],
  quantization : [
    low : [0/1, 10/0.7, 20/0.2, 30/0, 40/0],
    medium : [0/0, 10/0.5, 20/1, 30/0.5, 40/0],
    high : [0/0, 10/0, 20/0.3, 30/0.8, 40/1]
  ]
]).

fuzzyVarDef(humidity, [
  universe : [10, 30, 70, 90, 100],
  quantization : [
    low : [10/1, 30/0.7, 70/0.3, 90/0, 100/0],
    medium : [10/0, 30/1, 70/0.5, 90/0, 100],
    high : [10/0, 30/0, 70/0.6, 90/1, 100/1]
  ]
]).

fuzzyVarDef(evaporation_time, [
  universe : [1, 2, 3, 4, 5],
  quantization : [
    short: [1/1, 2/0.7, 3/0.2, 4/0, 5/0],
    medium : [1/0, 2/0.3, 3/1, 4/0.3, 5/0],
    long : [1/0, 2/0, 3/0.2, 4/0.8, 5/1]
  ],
  defuz : peak
]).
```

Quando as variáveis vagas do sistema são definidas através do predicado *fuzzyVarDef/2*, tal como exemplificado, é necessário executar um processo de tratamento das definições de variáveis vagas que as converte numa representação mais eficiente para a sua utilização pelo mecanismo de inferência. O pre-processamento de definições de variáveis, levado a cabo pelo procedimento *loadVariableDefinitions/0*, faz o seguinte:

Mantém uma lista com os nomes de todas as variáveis definidas, no facto *varList/1*, por exemplo varList([temperature, humidity, evaporation_time]). Esta lista é usada para consulta e para possibilitar a remoção de variáveis da memória.

O universo de discurso de uma variável é armazenado num facto do predicado *variableUniverse/2*, tal que variableUniverse(VarName, Universe) significa que o universo de discurso da variável com nome VarName é Universe. Um universo de discurso é representado através de uma lista, por exemplo [1, 2, 3, 4, 5].

A quantização vaga de uma variável vaga é composta pelos conjuntos vagos que representam os vários valores vagos que a variável pode tomar. A definição de cada um desses valores vagos é armazenada num facto *variableValueDef/3*, tal que variableValueDef(VarName, ValueName, ValueSet) significa que a variável vaga VarName pode tomar o valor vago ValueName, o qual é representado pelo conjunto vago ValueSet. Por exemplo, o facto variableValueDef(temperature, low, [0/1, 10/0.7, 20/0.2,

30/0, 40/0]) significa que a variável vaga *temperature* pode tomar o valor vago *low*, o qual é representado pelo conjunto vago [0/1, 10/0.7, 20/0.2, 30/0, 40/0].

Finalmente, o método de desfuzzificação de uma variável, caso seja especificado, é armazenado num facto do predicado *variableDefuzMethod/2*, tal que variableDefuzMethod(VarName, DefuzMethod) significa que o método de desfuzzificação da variável vaga VarName é DefuzMethod. Por exemplo, o facto variableDefuzMethod(evaporation_time, peak) significa que a variável vaga evaporation_time é desfuzzificada através do método dos máximos (peak). Se não for especificado um método de desfuzzificação de uma dada variável, e o FUZ-is tiver de a desfuzzificar, é usado o método do centro de massas (*centroid*).

O pré-processamento das definições de variáveis vagas (procedimento *loadVariableDefinitions/0*) dá origem a uma série de factos por cada variável: um facto com a lista dos nomes de todas as variáveis, um facto pelo universo do discurso, um facto por cada valor vago contido na quantização da variável e, por vezes, um facto com a especificação do método de desfuzzificação dessa variável. O seguinte exemplo mostra os factos criados pelo carregamento da definição da variável evaporation_time, a qual foi criada com o predicado *fuzzyVarDef/2* como se ilustra em cima:

varList([evaporation_time, ...]).

variableUniverse(evaporation_time, [1, 2, 3, 4, 5]).

variableValueDef(evaporation_time, short,

[1/1, 2/0.7, 3/0.2, 4/0, 5/0]).

variableValueDef(evaporation_time, medium,

[1/0, 2/0.3, 3/1, 4/0.3, 5/0]).

variableValueDef(evaporation_time, long,

[1/0, 2/0, 3/0.2, 4/0.8, 5/1]).

variableDefuzMethod(evaporation_time, peak).

Se a definição das variáveis voltar a ser carregada, os factos criados em carregamentos anteriores mantêm-se na memória do sistema. Na maior parte dos casos, isso deve ser evitado. Para isso, existe o procedimento *clearVariableDefinitions/0*, o qual pode ser usado para limpar todos esses factos criados no carregamento das variáveis vagas do sistema.

Poderá ser preferível e mais fácil enveredar pela primeira alternativa para a definição das variáveis vagas de um sistema. Em vez de recorrer a factos *fuzzyVarDef/2*, especificar diretamente os factos *variableUniverse/2*, *variableValueDef/3* e *variableDefuzMethod/2* e não usar os procedimentos *loadVariableDefinitions/0* e *clearVariableDefinitions/0*.

Desfuzzificação de variáveis intermédias

Além das variáveis de saída, por vezes é necessário saber o valor exato (*"crisp value"*) de uma variável intermédia, o qual é usado posteriormente como input de uma regra que vai ser usada de seguida. Por exemplo, se um conjunto de regras é usado para determinar o valor da gratificação a dar a um empregado de mesa, em função da qualidade da comida e da qualidade do serviço; e a qualidade do serviço é determinada, em função do tempo de espera e da simpatia, por outro conjunto de regras, a variável que representa a qualidade de serviço é uma variável intermédia cujo valor exato tem de ser determinado pelo sistema antes de poder ser usada para determinar o valor da gratificação. É necessário usar um método de desfuzzificação para essas variáveis intermédias cujo valor exato tem de ser conhecido.

Quando o método de desfuzzificação não é especificado explicitamente, o sistema usa o método do *centro de massas*.

Quando o valor exato de uma de uma variável intermédia, obtido através da sua desfuzzificação é um valor diferente dos valores do universo de discurso dessa variável, embora no mesmo intervalo de variação, o sistema faz uma interpolação simples do grau de pertença desse valor exato no conjunto vago considerado.

Por exemplo, imagine-se que o universo de discurso da qualidade do serviço é o conjunto {1, 2, 3, 4, 5}, mas o valor exato computado pelo sistema para essa variável é de 1.67, o qual resulta de uma desfuzzificação, usando o método especificado ou o método usado na ausência de especificação.

Neste exemplo, uma das regras que usa o valor da qualidade de serviço tem uma condição que inclui a proposição atómica (serviço is medio). Supondo, por fim, que o conjunto vago que representa a qualidade de serviço média é {1/0, 2/0.5, 3/1, 4/0.5, 5/0}, o valor de verdade da proposição (serviço is medio) para o valor exato do serviço 1.67 é determinado por interpolação. Como 1.67 se situa entre os valores 1 e 2 do universo de discurso, o valor de pertença será um valor entre 0 e 0.5, o qual é calculado da seguinte maneira:

$0 \times (1 - (1.67-1)/(2-1)) + 0.5 \times (1 - (2-1.67)/(2-1)) = 0.335$

O predicado do *FUZ-is* que faz esta interpolação chamado *guessValue/5* e tem a definição que se segue:

```
guessValue(V, Set, Lesser/T1, Greater/T2, Value):-
  Max is Greater - Lesser,
  LDif is V - Lesser,
  GDif is Greater - V,
  RLDif is 1 - LDif / Max,
  RGDif is 1 - GDif / Max,
  Value is T1 * RLDif + T2 * RGDif.
```

Esta é uma interpolação linear, mas poderia ter sido usada um método de interpolação diferente.

6.2 Regras vagas

O FUZ-is possibilita a escrita de regras condição-conclusão. A condição de uma regra pode ser um literal (proposição atómica ou proposição atómica negada) ou qualquer combinação de conjunções, disjunções e negações. A conclusão de uma regra pode ser apenas um literal.

O seguinte exemplo ilustra a forma de escrever regras.

```
% Determinação do tempo de evaporação

if (temperature is low and humidity is high)
then (evaporation_time is long).

if (temperature is medium and humidity is medium)
then (evaporation_time is medium).

if (temperature is high and humidity is low)
then (evaporation_time is short).
```

As regras de um sistema baseado em lógica vaga podem ser encadeadas, umas nas outras. Isto é, parte ou a totalidade da condição de uma ou mais regras podem depender do valor de variáveis determinadas nas conclusões de outras regras. No exemplo que se segue, as três primeiras regras determinam a qualidade de serviço, a qual vai ser usada nas segundas três regras.

% Determinação da qualidade de serviço

```
if (tempo_espera is elevado and simpatia is fraca)
then (servico is mau).

if (tempo_espera is medio and simpatia is media)
then (servico is medio).

if (tempo_espera is baixo and simpatia is grande)
then (servico is bom).

% Determinação da gratificação

if (comida is fraca and servico is mau) then (gorjeta is pequena).

if (comida is media and servico is medio) then (gorjeta is media).

if (comida is boa and servico is bom) then (gorjeta is grande).
```

Quando se pede ao sistema para calcular a gratificação a dar ao empregado de mesa do restaurante, em função do tempo de espera, da simpatia e da qualidade da comida, são usadas as regras do segundo grupo, as quais dependem da qualidade do serviço. Nesta altura, são as regras do primeiro grupo que intervêm na determinação da qualidade do serviço. O valor exato da qualidade de serviço, o qual é necessário como input do segundo grupo de regras, é determinado pelo *FUZ-is* mediante a utilização de um método de

desfuzzificação. Se não for especificado o método de desfuzzificação da variável, será usado o método do centro de massas, o qual é assumido por omissão (ver secção 6.1).

6.3 Configuração

O FUZ-is é bastante flexível em termos das possibilidades de escolhas de métodos para as mais diversas fases do processo de inferência:

- Podem escolher-se os métodos usados para determinar o grau de verdade de conjunções, disjunções e negações.
- Pode escolher-se o método usado para determinar o conjunto vago da conclusão de uma regra, dado o grau de verdade da condição dessa regra.
- Pode escolher-se o método usado para combinar contribuições de todas as regras para uma dada variável.
- Pode escolher-se o método usado para desfuzzificar um conjunto vago.

O FUZ-is oferece várias alternativas para a maioria das escolhas mencionadas. Por exemplo, os métodos disponíveis para determinar o grau de verdade de uma conjunção são o método dos mínimos, o método do produto e o método do produto limitado. As outras escolhas dispõem igualmente de várias alternativas.

Se, em qualquer escolha, o leque de opções disponíveis não for adequado face à natureza e objetivos do sistema que se pretende criar, é possível programar novos métodos desde que a sua interface seja a mesma da dos métodos disponíveis para essa escolha.

O método usado para desfuzzificar um conjunto vago é especificado juntamente com a definição das variáveis vagas. Os outros métodos são descritos seguidamente:

Métodos para determinar o grau de verdade da conjunção

A escolha do método para determinar o grau de verdade da conjunção é feita através de um facto do predicado *conjunctionTruthMethod/1*. conjunctionTruthMethod(Method) significa que Method é o método escolhido para a conjunção.

Existem os seguintes métodos com as designações correspondentes:

Método do mínimo: *minMethod*

Método do produto: *productMethod*

Método do produto limitado: *boundedProdMethod*

Para escolher o método do mínimo, por exemplo, teríamos de ter o facto

```
conjunctionTruthMethod(minMethod).
```

Se a implementação de um método diferente for desejada, bastará criar um predicado Prolog com a seguinte interface:

<Method Name>(+TruthValue1, +TruthValue2, ?Result)

TruthValue1 representa o valor de verdade de uma das duas proposições que são combinadas na conjunção; *TruthValue2* é o valor de verdade da outra proposição. Tanto um como outro são valores reais entre 0 e 1, inclusive e serão obrigatoriamente instanciados. *Result* é o valor de verdade da conjunção e será determinado pelo predicado.

Métodos para determinar o grau de verdade da disjunção

A escolha do método para determinar o grau de verdade da disjunção é feita através de um facto do predicado *disjunctionTruthMethod/1*. disjunctionTruthMethod(Method) significa que Method é o método escolhido para a disjunção.

Existem os seguintes métodos com as designações correspondentes:

Método do máximo: *maxMethod*

Método do Probor (Probabilistic OR): proborMethod

Método da soma limitada: *additionMethod*

Para escolher o método do máximo, por exemplo, teríamos de ter o facto

```
disjunctionTruthMethod(maxMethod).
```

Se a implementação de um método diferente for desejada, bastará criar um predicado Prolog com a seguinte interface:

<Method Name>(+TruthValue1, +TruthValue2, ?Result)

TruthValue1 representa o valor de verdade de uma das duas proposições que são combinadas na conjunção; *TruthValue2* é o valor de verdade da outra proposição. Tanto um como outro são valores reais entre 0 e 1, inclusive e serão obrigatoriamente instanciados. *Result* é o valor de verdade da disjunção e será determinado pelo predicado.

Método para determinar o grau de verdade da negação

A escolha do método para determinar o grau de verdade da negação é feita através de um facto do predicado *negationTruthMethod/1*. negationTruthMethod(Method) significa que Method é o método escolhido para a negação.

Existe apenas um método com a seguinte designação:

Método do complementar: *complement*

Para escolher o método do máximo, que é o único que se pode escolher atualmente, teríamos de ter o facto

```
negationTruthMethod(complement).
```

Se a implementação de um método diferente for desejada, bastará criar um predicado Prolog com a seguinte interface:

<Method Name>(+TruthValue, ?Result)

TruthValue representa o valor de verdade da proposição que é negada. TruthValue é um real entre 0 e 1, inclusive e tem obrigatoriamente de ser instanciado. *Result* é o valor de verdade da negação.

Método para determinar a contribuição produzida por uma regra

Depois de computar o grau de verdade da condição de uma regra, é necessário determinar a contribuição dessa regra para o valor final da variável em que estamos interessados. A contribuição de uma regra para o valor de uma variável é um conjunto vago que se obtém, sabendo o grau de verdade da condição, através do método selecionado.

A escolha do método para computar o conjunto vago que representa a contribuição da regra para o valor da variável é feita através de um facto do predicado *condition2conclusionPropagationMethod/1*. condition2conclusionPropagationMethod(Method) significa que Method é o método escolhido para calcular a contribuição da regra com base no valor de verdade da condição da regra.

Existem os seguintes métodos com as designações correspondentes:

Método truncar: *truncate*

Método escalar: *shrink*

Para escolher o método de escalar, por exemplo, teríamos de ter o facto

```
condition2conclusionPropagationMethod(shrink).
```

Se a implementação de um método diferente for desejada, bastará criar um predicado Prolog com a seguinte interface:

<Method Name>(+ConditionTruthValue, +ConclusionSet, ?Result)

ConditionTruthValue representa o valor de verdade da condição da regra. *ConclusionSet* é a definição do conjunto vago que representa o valor vago da proposição atómica que aparece na conclusão da regra. *ConditionTruthValue* é um real entre 0 e 1, inclusíve e tem obrigatoriamente de estar instanciada. *ConclusionSet* é uma lista de pares (valor/pertença) e tem obrigatoriamente de estar instanciada. Nestes pares, *valor* é um elemento do universo de discurso da variável; *pertença* é o grau de pertença de valor no conjunto vago que representa o valor vago da variável da proposição que surge na conclusão da regra.

Result é o conjunto vago que representa a contribuição da regra e será determinado pelo predicado. *Result* é uma lista de pares (valor/pertença).

Método para combinar todas as contribuições das regras para uma variável

As várias contribuições para uma dada variável são representadas pelos conjuntos vagos produzidos pelas várias regras que contribuem para essa variável. O método para combinar as várias contribuições tem fundamentalmente que produzir um conjunto vago que resulta da combinação dos vários conjuntos vagos que resultam das várias regras que contribuem para a variável em causa. É frequente designar esse método de combinação por *else link*.

A escolha do *else-link* usado para combinar as várias contribuições para uma variável é feita através de um facto do predicado *else_linkMethod/1*. else_linkMethod(Method) significa que Method é o método escolhido para a combinação das contribuições.

No FUZ-is existem os seguintes métodos com as designações correspondentes:

Or-link: *or_link*

And-link: *and_link*

Probor (Probabilistic OR): *probor_link*

Para escolher o *or-link*, por exemplo, teríamos de ter o facto

```
else_linkMethod(or_link).
```

Se a implementação de um método diferente for desejada, bastará criar um predicado Prolog com a seguinte interface:

<Method Name>(+Contributions, ?Result)

Contributions é uma lista de conjuntos vagos, cada um dos quais representa uma das contribuições para a variável em causa. Cada conjunto vago é representado por uma lista de pares (*valor*/*pertença*). Nestes pares, *valor* é um elemento do universo de discurso da variável; *pertença* é o grau de pertença de valor ao conjunto vago. *Contributions* tem de ser obrigatoriamente instanciada. *Result* é o conjunto vago que representa a combinação das várias contribuições para a variável em causa. *Result*, que é computado pelo predicado, é uma lista de pares (valor/pertença). Nestes pares, *valor* é um elemento do universo de discurso da variável; *pertença* é o grau de pertença de valor ao conjunto vago.

6.4 Utilização do sistema baseado em lógica vaga

Nesta secção apresentam-se essencialmente dois tópicos ligados à utilização do *FUZ-is* para fazer sistemas baseados em conhecimento:

- Predicados de interação com o sistema baseado em conhecimento
- O desenvolvimento das bases de conhecimentos e carregamento do FUZ-is

Interação com o mecanismo de inferência do *FUZ-is*

FUZ-is tem dois predicados que podem ser usados para implementar a interação com o sistema. Esses dois predicados servem para se fazer perguntas ao sistema, as quais envolvem a utilização de inferência. O FUZ-is implementa o método de inferência de valores exatos. Tanto no caso de um predicado, como no do outro, o sistema tem de ter acesso aos valores exatos das variáveis de input e calcula o valor da variável de output especificada.

Os predicados de inferência do FUZ-is são *fuzis/2* e *fuzis/3. fuzis/2* recebe o nome de uma variável vaga e calcula o seu valore exato usando as regras e as definições dos valores vagos da base de conhecimento. Os valores exatos das variáveis de input são obtidos de factos *input/2*, os quais mantêm o nome da variável de input e o seu valor. *fuzis/3* recebe uma lista com os valores exatos das variáveis de input, o nome de uma variável vaga e calcula o seu valor exato. A lista de valores exatos das variáveis de input é constituída por pares (variable = value), em que *variable* é o nome da variável de input e *value* é o seu valor exato.

Dado que o FUZ-is é totalmente implementado em Prolog, *fuzzis/2* e *fuzzis/3* são predicados normais de Prolog, podendo ser usados em qualquer situação em que um outro predicado possa ser usado,

nomeadamente na linha de comando do interpretador de Prolog ou na definição de outro predicado qualquer.

Suponhamos que temos um sistema de regras baseado em lógica vaga para determinar o valor da gratificação a dar ao empregado de mesa num restaurante. As variáveis de input desse sistema são o tempo de espera (tempo_espera), a simpatia (simpatia) e a qualidade da comida (comida). A variável de output é o valor da gratificação (gorjeta).

Nesse sistema podemos usar *fuzis/3* e *fuzis/2* para obter o valor da gratificação sugerida pelo sistema. Comecemos por usar *fuzis/3* na linha de comando.

```
?- fuzis([tempo_espera=10,simpatia=4,comida=4], gorjeta, Gorjeta).

Gorjeta = 4.5;

no
```

fuzis/3 pode também ser usado na definição de outro predicado, por exemplo no predicado *gorjeta/4*:

```
gorjeta(Tempo, Simpatia, Comida, Gorjeta):-
   fuzis([tempo_espera=Tempo, simpatia=Simpatia, comida=Comida],
    gorjeta, Gorjeta).

?- gorjeta(10, 4, 4, Gorjeta).

Gorjeta = 4.5;

no
```

Em vez de *fuzis/3*, pode ser usado o predicado *fuzis/2* o qual tem a particularidade de não receber os valores das variáveis vagas de input como argumento. Em vez disso, os valores das variáveis de input são previamente armazenadas em factos do predicado *input/2*. As interacções exemplificadas, na linha de comando do interpretador Prolog, usando tanto o predicado *fuzis/3* como *gorjeta/4* podem ser replicadas usando o predicado *fuzis/2*. Primeiro criam-se os factos *input/2* com os valores exatos das variáveis de input: tempo_espera, simpatia e comida:

```
input(tempo_espera, 10).

input(simpatia, 4).

input(comida, 4).
```

Depois destes factos terem sido criados, ou através de *asserts* na linha de comando do interpretador, ou através do carregamento de um ficheiro Prolog com esses factos, ou de qualquer outra forma, o *fuzis/2* pode ser usado como na seguinte interação:

```
?- fuzis(gorjeta, Gorjeta).

Gorjeta = 4.5;

no
```

A utilização de *fuzis/2* em vez de *fuzis/3* é mais adequada em situações em que há muitas variáveis de input, não sendo agradável especificá-las todas na linha de comando ou em situações em que é necessário armazenar os valores das variáveis de input.

Desenvolvimento de SBCs e carregamento do *FUZ-is*

O FUZ-is é implementado em Prolog. O seu código distribui-se por quatro ficheiros: FUZ-is.pl, Inference.pl, Methods.pl e Vardef.pl.

Inference.pl contém o mecanismo de inferência do *FUZ-is*.

Methods.pl contém a definição dos vários métodos usados para as várias operações do mecanismo de inferência (ver secção 6.3)

Vardef.pl contém os procedimentos *loadVariableDefinitions/0* e *clearVariableDefinitions/0* usados para carregar e limpar as definições de variáveis vagas (ver secção 6.1).

fuz-is.pl é um ficheiro que se limita a definir o *pathname* da directoria onde o código fonte do FUZ-is está localizado e carrega os outros ficheiros que pertencem à distribuição do sistema.

Para desenvolver um SBC, usando o FUZ-is, é necessário o seguinte:

Um interpretador de Prolog onde executar o sistema:

Editar o ficheiro fuz-is.pl e alterar o *pathname* da directoria onde o código do FUZ-is está localizado. Isso faz-se pela alteração do facto *fuzis_home_dir/1*.

Pelo menos, um ficheiro onde o conhecimento da base de conhecimentos do sistema deve ser representado.

O ficheiro com a base de conhecimentos deve começar por um comando para carregar o ficheiro *fuz-is.pl* o qual se encarregará de mandar carregar os outros ficheiros Prolog que constituem o *FUZ-is*. Assumindo que a directoria onde está localizado o FUZ-is tem o pathname C:\Prolog\FL, o comando necessário é o seguinte:

:- ensure_loaded('C:/Prolog/FL/fuz-is.pl')[4].

Se o interpretador de Prolog usado não reconhecer o comando *ensure_loaded/1*, deve-se tentar o comando *reconsult/1* em vez de *ensure_loaded/1*:

:- reconsult('C:/Prolog/FL/fuz-is.pl').

Depois deste comando, é necessário criar as regras vagas (ver secção 6.2), escrever as definições das variáveis vagas (ver secção 6.1), fazer a configuração do sistema (ver secção 6.3), e definir a interface com o sistema (ver secção 6.4).

Depois de criada a base de conhecimentos, o ficheiro deve ser carregado no interpretador Prolog. Depois disso, se o predicado *fuzzyVariableDef/2* tiver sido usado, deve carregar-se a definição das variáveis:

```
?- loadVariableDefinitions.

yes
```

Depois das variáveis terem sido carregadas, pode usar-se o sistema através da interface definida (*fuzis/3* e *fuzis/2*).

Se necessário limpar as representações internas das variáveis vagas, criadas pelo procedimento *loadVariableDefinitions/0*, pode usar-se o procedimento *clearVariableDefinitions/0*.

7 FLINT: Inferência com valores exatos

[Esta ferramenta deixou de ser utilizada nas aulas de Tecnologias para Sistemas Inteligentes a partir de 2008-2009 inclusive]

O *Win-Prolog* da LPA dispõe de um sistema de representação e raciocínio para o desenvolvimento de Sistemas Baseados em Lógica Vaga. Este sistema chama-se FLINT (Fuzzy Logic Interpreter). Esta secção explica o conjunto mínimo de conhecimentos para a utilização do FLINT.

O FLINT é um módulo do *Win-Prolog* que tem de ser carregado no interpretador para que o possamos usar. O FLINT permite duas funcionalidades: definir uma base de conhecimentos constituída por regras baseadas em lógica vaga e pelos conjuntos vagos necessários para representar todos os conceitos imprecisos referidos nas regras; e fazer inferência vaga usando as regras e conjuntos definidos na base de conhecimentos. Existem diversos predicados Prolog que podem ser usados no processo de inferência vaga. Usando estes predicados podemos criar aplicações em Prolog que recorrem a sistemas baseados em lógica vaga quando isso for desejável.

As próximas subsecções descrevem estes três assuntos. A subsecção 7.1 descreve como se carrega o FLINT no interpretador. A subsecção 0 explica a criação de uma base de conhecimentos formada por regras vagas e por conjuntos vagos. A subsecção 7.3 descreve os predicados Prolog usados para efetuar inferência com as regras e conjuntos representados na base de conhecimento do sistema. Estes predicados podem ser integrados em qualquer programa Prolog o que possibilita o desenvolvimento de aplicações em Prolog que recorrem, se desejável, à lógica vaga. Finalmente, a subsecção 0 apresenta o suporte do FLINT para facilitar o seguimento do funcionamento de sistemas.

---

[4] Há sistemas de Prolog, por exemplo o Win-Prolog da LPA (Logic Programming Associates) que só reconhecem a sintaxe usada no Windows para especificar *pathnames*:

```
:- ensure_loaded('C:\Prolog\FL\fuz-is.pl').
```

7.1 Carregamento do FLINT

Para usar o FLINT é necessário carregá-lo no interpretador de Prolog. O predicado *ensure_loaded/1* pode ser usado para efetuar o referido carregamento, usando o seguinte comando:

```
?- ensure_loaded(system(flint)).

yes
```

O predicado *ensure_loaded/1* é o predicado standard para carregar módulos no interpretador. Esses módulos serão ficheiros com código fonte Prolog (extensão ".pl") ou ficheiros compilados (com extensão ".pc"), como é o caso do FLINT.

ensure_loaded/1 tem como argumento a especificação do módulo a carregar. Essa especificação poderá ser um *pathname* absoluto ou relativo, ou então a especificação de módulos pertencentes ao próprio LPA, como é o caso do FLINT.

A distribuição do *Win-Prolog* está organizada em várias directorias, entre as quais as directorias System e Examples. A directoria Examples contém ficheiros com exemplos de vários tipos. Uma das subdirectorias de Examples é a directoria FLINT a qual contém exemplos de utilização do FLINT. Para especificar módulos do LPA, tem que se indicar um functor com o nome da directoria da distribuição do Win-Prolog que contém o módulo que pretendemos carregar. No caso do comando apresentado, é usado o functor "System" porque o FLINT está nessa directoria.

Por exemplo, o comando

?- ensure_loaded(examples('flint\turbine.pl')).

yes

carrega o ficheiro Turbine.pl com um exemplo completo de utilização do FLINT.

Uma vez carregado o FLINT, podemos carregar ficheiros com bases de conhecimentos com regras e conjuntos vagos e podemos fazer inferência com essas bases de conhecimentos.

O FLINT pode ser usado na versão 4.6.0 do *Win-Prolog*. Ele existia em versões anteriores como por exemplo, a 4.3.2, mas não funcionava corretamente.

Base de Conhecimentos com Regras e Conjuntos Vagos

Uma base de conhecimentos FLINT, como todas as bases de conhecimentos baseadas em lógica vaga, é constituída por regras e por conjuntos vagos.

Existem quatro predicados que permitem a criação de bases de conhecimentos em FLINT: *fuzzy_variable/1*, *fuzzy_hedge/2*, *uncertainty_rule/1* e *fuzzy_matrix/1*.

O predicado *fuzzy_variable/1* serve para definir o universo de discurso e a quantização vaga de uma variável vaga a usar em regras vagas. A quantização vaga de uma variável é o conjunto de valores vagos que ela pode tomar. O predicado *fuzzy_hedge/2* serve para definir modificadores. Um modificador pode ser usado em qualquer regra da base de conhecimentos, quer na condição como na conclusão. O predicado *uncertainty_rule/1* serve para definir uma regra vaga. Finalmente, o predicado *fuzzy_matrix/1* serve para definir, de uma só vez, uma matriz de regras vagas, todas elas com formato semelhante.

Variáveis vagas

Exemplo.

```
fuzzy_variable(speed):-
   [0, 200];
   slow, \, curved(2), [0, 30];
   medium, /\, linear, [10, 30, 50];
   fast, /, curved(0.5), [40, 70];
   centroid.
```

A variável vaga definida neste exemplo chama-se *speed* e pode variar no intervalo 0 a 200, inclusive. Os valores vagos que a variável *speed* pode tomar (a sua quantização vaga) são *slow*, *medium* e *fast* cuja definição é feita também no predicado *fuzzy_variable/1*.

O símbolo \ na definição de *slow* indica que se trata de um conjunto vago que desce a partir de um determinado ponto. A expressão curved(2) significa que essa descida tem uma forma curva com um

determinado tipo de curvatura. O intervalo [0, 30] na definição de *slow* significa que se trata de um conjunto cujos graus de pertença começam a descer a partir de uma velocidade igual a 0 e que atingem o valor 0 para uma velocidade igual a 30.

A definição do conjunto *medium* especifica que se trata de uma função de pertença constituída por segmentos de reta (linear) que começa a subir a partir da velocidade com valor 10, que atinge o maior grau de pertença (1) quando a velocidade toma o valor 30 e que volta a descer daí em diante até voltar a 0 quando a velocidade é igual a 50.

A definição do conjunto *fast* especifica uma função de pertença constituída por linhas curvas com um dado tipo de curvatura (curved(0.5)), a qual é de tipo diferente da curvatura do conjunto *slow*, que começa a subir quando a velocidade é 40 e atinge o grau de pertença máximo (1) quando a velocidade é 70. Note-se que o grau de pertença do conjunto *fast* é 0 para valores da velocidade entre 0 (limite inferior do universo de discurso até 40, e é 1 de 70 até 200 (limite superior do universo de discurso).

Finalmente, a desfuzzificação da variável *speed* é feita usando o método centroide (centro de massas). Alternativamente poderia ter sido especificado o método dos máximos o que seria indicado pelo átomo *peak* em vez de *centroid*. Não é necessário indicar o método de desfuzzificação para variáveis que não necessitam ser desfuzzificadas.

A Tabela 6 descreve os símbolos que se podem usar para definir a forma geral das funções de pertença. A Tabela 7 apresenta os tipos de curvatura definidos pelo parâmetro do functor *curved*.

Símbolo	Parâmetros	Descrição
\	[A, B]	Função de pertença cujos graus de pertença são 1 para todos os valores desde o limite inferior do universo de discurso até A. Começa a descer em A, atingindo o grau de pertença 0 no valor B. Mantém-se 0 desde B até ao limite superior do universo discurso. A descida é uma linha reta ou curva de acordo com a especificação.
/	[A, B]	Função de pertença cujos graus de pertença são 0 para todos os valores desde o limite inferior do universo de discurso até ao valor A. Começa a subir em A e atinge o grau de pertença 1 em B. Depois mantêm-se igual a 1 até ao limite superior do universo de discurso. A subida é uma linha reta ou curva de acordo com a especificação
/\	[A, B, C]	Função de pertença cujos graus de pertença são zero para valores iguais ou menores que A ou iguais ou maiores que C. Começa a subir em A atingindo o valor 1 em B, a partir do que volta a descer até atingir 0 em C. As subida e descida são linhas retas ou curvas de acordo com a especificação.
\/	[A, B, C]	O simétrico da anterior.
/-\	[A, B, C, D]	Neste caso os graus de pertença são zero para valores menores ou iguais a A e para valores maiores ou iguais a D. A função de pertença inicia a sua subida em A, atingindo o valor 1 em B. Mantém-se com o valor 1 até C, altura em que começa a descer até atingir de novo o valor 0 em D. As subida e descida são linhas retas ou curvas de acordo com a especificação
\-/	[A, B, C, D]	O simétrico da anterior.
?	[V_1/μ_1, …, V_m/μ_m]	Função de pertença em que o utilizador fornece os graus de pertença μ_1, …, μ_m para os valores V_1, …, V_m. Os valores V_1, …, V_m têm obrigatoriamente que ser dados por ordem estritamente crescente. As linhas que unem os pontos dados são segmentos de reta ou curvas conforme a especificação.

Tabela 6 – Formas Gerais de Funções de Pertença

A seguinte expressão mostra um exemplo de uma função de pertença totalmente especificada pelo programador: `extreme, ?, linear, [60/1, 65/0.75, 70/0.4, 85/0.25, 90/0]`. Neste caso os pontos especificados são unidos por segmentos de reta.

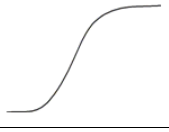
Parâmetro de curvatura (argumento de curved(P))	Formato da Curvatura	
< 1		
1		
> 1		

Tabela 7 – Curvatura as Funções de Pertença

Como se pode ver na Tabela 7, a especificação *curved(1)* é igual à especificação linear.

Modificadores

Exemplos

fuzzy_hedge(very, power(2)).

fuzzy_hedge(slightly, power(0.5)).

Nos dois exemplos apresentados, *very* e *slightly* são átomos que designam os modificadores (*hedges*) definidos. *power(2)* é a expressão que define o modificador *very*. Isto significa que o modificador *very* cria um novo conjunto vago cujos graus de pertença são o quadrado dos valores da função de pertença do conjunto a que se aplica. *power(0.5)* é a expressão que define o modificador *slightly*. Isto significa que o operador *slightly* cria um novo conjunto vago cujos graus de pertença são a raiz quadrada do conjunto a que se aplicar. Por exemplo, se *fast* for um conjunto vago, *very fast* é o conjunto cujos graus de pertença são o quadrado dos graus de pertença de *fast*; e *slightly fast* é o conjunto vago cujos graus de pertença são a raiz quadrada dos graus de pertença de *fast*.

Qualquer modificador definido se pode aplicar a qualquer conjunto vago.

Regras vagas

Sintaxe:

```
uncertainty_rule(<RuleName>):-
   if <Condittion> then <Conclusion>
   else <Conclusion>.
```

ou

```
uncertainty_rule(<RuleName>):-
   if <Condittion> then <Conclusion>.
```

O FLINT permite a utilização de regras com *else*, mas não falaremos delas aqui.

As condições das regras podem conter os operadores lógicos **and**, **or** e **not**. Mas chama-se a atenção para o facto lamentável de o operador **not** não ser a negação mas sim o operador de complemento de um conjunto, isto é, o **not** aplica-se a um conjunto e não a uma proposição.

Exemplo de condição:

(temperature is not very high) or (humidity is medium)

Exemplo de regra

```
uncertainty_rule(risk1):-
   if duration is long and staffing is not large then risk is high.
```

7.2 Matrizes de regras vagas

O predicado *uncertainty_rule/1* serve para definir uma regra de cada vez, mas por vezes seria desejável e é possível definir uma matriz de regras. Uma matriz de regras representa uma tabela em que, para cada condição se define a conclusão. Para se definir uma matriz de regras, é necessário que todas as regras tenham condições exatamente com o mesmo número de condições elementares, incidindo sobre as mesmas variáveis, todas elas ligadas com operador **and**. A Tabela 8 mostra o exemplo de uma matriz de regras. Dada uma duração e um custo (nas duas primeiras colunas), pode obter-se o risco na terceira coluna.

Duration	Cost	Risk
Long	High	very High
Short	Low	Low

Tabela 8 – Matriz de Regras

Claro que apenas para duas regras, há pouca vantagem em definir uma matriz de regras, mas para mais regras, as vantagens são cada vez maiores.

A Tabela 8 pode ser representada usando as duas seguintes regras:

```
uncertainty_rule(risk1):-
   if duration is long and cost is high then risk is very high.

uncertainty_rule(risk1):-
   if duration is short and cost is low then risk is low.
```

Em vez de definir duas regras, uma de cada vez, pode definir-se uma matriz com as duas regras. O predicado *fuzzy_matrix/1* é usado para definir matrizes de regras.

Exemplo

```
fuzzy_matrix(risk_rules):-
   duration  * cost -> risk;
   long      * high -> very high;
   short     * low  -> low.
```

risk_rules é o nome da matriz. A primeira linha da matriz especifica as variáveis envolvidas na matriz: *duration*, *cost* e *risk*. As linhas seguintes concretizam a tabela para valores específicos das variáveis de entrada e de saída.

7.3 Inferência

Esta secção apresenta os métodos usados pelo FLINT para calcular o grau de verdade das Conjunções e das Disjunções em termos dos graus de verdade das suas proposições elementares; e descreve os predicados usados para efetuar inferência.

Valor de verdade das conjunções e das disjunções

A Tabela 9 mostra os métodos alternativos usados pelo FLINT para determinar os valores de verdade da conjunção e da disjunção em termos dos valores de verdade das suas proposições constituintes.

P and Q	Minimum (default)	$\text{Min}(\mu_P(u), \mu_Q(v))$
	Product	$\mu_P(u) \times \mu_Q(v)$
	Truncate	$\text{Max}(\mu_P(u)+\mu_Q(v)-1, 0)$
P or Q	Maximum (default)	$\text{Max}(\mu_P(u), \mu_Q(v))$
	Strengthen	$\mu_P(u) + \mu_Q(v) \times (1-\mu_P(u))$
	Addition	$\text{Min}(\mu_P(u)+\mu_Q(v), 1)$

Tabela 9 – Funções de Verdade da Conjunção e da Disjunção

A escolha dos métodos enumerados na Tabela 9 é feita no predicado *fuzzy_propagate/4* (ver secção 0).

Predicados usados na inferência

Apesar do FLINT dispor de inúmeros predicados que podem ser usados para efetuar inferência, aqui apenas são apresentados três que permitem fazer inferência e criar aplicações em Prolog que recorrem a inferência vaga: *fuzzy_variable_value/2*, *fuzzy_reset_membership/1*, *fuzzy_propagate/4*, e *fuzzy_propagate/1*. Mais predicados podem ser consultados na documentação sobre FLINT da LPA.

fuzzy_variable_value/2 é usado para estabelecer ou para obter o valor exato de uma variável. Em caso da obtenção do valor de uma variável, usa-se o método de desfuzzificação definido no predicado *fuzzy_variable/1*.

fuzzy_reset_membership/1 é usado para reiniciar o valor vago de uma variável vaga. Usa-se antes de determinar o valor da variável vaga de saída.

fuzzy_propagate/4 é usado para aplicar as regras vagas de modo a determinar os valores vagos da variável de saída. Este predicado permite escolher os métodos usados para determinar os valores de verdade da conjunção, da disjunção e do complemento de um conjunto. *fuzzy_propagate/1* define-se à custa de *fuzzy_propagate/4*, com os métodos usados por omissão para a disjunção, a conjunção e o complemento.

Sintaxe

fuzzy_variable_value(<Fuzzy Variable>, <Crisp Value>)

Atribui o valor exato <Crisp Value> à variável vaga <Fuzzy Variable>, ou desfuzzifica o valor da variável, usando o método especificado em *fuzzy_variable/1*, unificando-o com a variável <Crisp Value>.

fuzzy_reset_membership(<Fuzzy Variable>)

Reinicia o valor vago da variável <Fuzzy Variable>.

fuzzy_propagate(<And Method>, <Or Method>, <Compliment Method>, <Rules>)

Utilizando os métodos <And Method>, <Or Method> e <Compliment Method>, aplica as regras <Rules> para determinar as contribuições de cada regra para a variável de saída. <Rules> é uma lista contendo os nomes das regras individuais e/ou das matrizes de regras a usar.

```
fuzzy_propagate(Rules):-
   fuzzy_propagate(minimum, maximum, complement, Rules).
```

Exemplo

Consideremos um exemplo em que temos um sistema para determinar o tempo de evaporação de uma determinada quantidade fixa de água, em termos da temperatura ambiente e da humidade relativa. Para isso criamos um programa Prolog chamado *evaporation_time/3* que recebe o valor da temperatura e o valor da humidade e devolve o tempo de evaporação. Este predicado recorre ao sistema baseado em lógica vaga criado especialmente para o efeito.

As regras usadas para determinar o valor do tempo de evaporação foram definidas na matriz de regras chamada *time_rules*. As variáveis de entrada do sistema chamam-se *temperature* e *humidity*. A variável de saída chama-se *time*.

```
evaporation_time(Temperature, Humidity, Time):-
   fuzzy_reset_membership(time),
   fuzzy_variable_value(temperature, Temperature),
   fuzzy_variable_value(humidity, Humidity),
   fuzzy_propagate([time_rules]),
   fuzzy_variable_value(time, Time).
```

O predicado começa por reiniciar o valor da variável de saída (*time*). Depois, atribui os valores das variáveis Prolog *Temperature* e *Humidity* às variáveis de entrada do sistema baseado em lógica vaga *temperature* e *humidity*. Depois usa as regras da matriz *time_rules* e finalmente obtém o valor da variável de saída *time*, unificando o seu valor com a variável Prolog *Time*.

Este predicado usa-se como qualquer outro predicado Prolog, por exemplo:

```
?- evaporation_time(25, 70, Time).
Time = 3
yes
```

O predicado *evaporation_time/3* pode ser usado em qualquer outro programa Prolog, o que permite misturar programas comuns em Prolog com sistemas desenvolvidos em FLINT.

Trace de Sistemas

O funcinamento de sistemas desenvolvidos em FLINT pode ser examinado através dos predicados *uncertainty_trace/0* e *uncertainty_notrace/0*.

Estes dois predicados iniciam e terminam o modo de seguimento (*tracing*) de sistemas FLINT. Seguidamente apresenta-se uma interação com o sistema para determinação do valor da gratificação em função da qualidade do serviço e da comida, tendo sido activado o modo de seguimento do funcionamento de sistemas; seguida de uma interação idêntica mas estando desactivado o modo de seguimento.

Como podemos constatar, o modo de seguimento permite observar a fase de fuzzificação dos inputs; a aplicação dos operadores lógicos da condição de de cada regra; e a desfuzzificação dos outputs.

```
?- uncertainty_trace.
yes
```

```
?- findtip(2, 4, Tip).
Mem.  : FUZZIFY    : food = 2
Mem.  : UPDATE     : (food is poorFood) = 0
Mem.  : UPDATE     : (food is goodFood) = 0.5
Mem.  : FUZZIFY    : service = 4
Mem.  : UPDATE     : (service is poorService) = 0
Mem.  : UPDATE     : (service is goodService) = 1
Mem.  : UPDATE     : (service is excellentService) = 0
Mem.  : TRY        : r1
Mem.  : LOOKUP     : (food is goodFood) = 0.5
Mem.  : LOOKUP     : (service is excellentService) = 0
Mem.  : AND        : 0.5 + 0 -> 0
Mem.  : UPDATE     : (tip is generous) = 0
Mem.  : FIRED      : r1
Mem.  : TRY        : r2
Mem.  : LOOKUP     : (food is poorFood) = 0
Mem.  : LOOKUP     : (service is poorService) = 0
Mem.  : OR         : 0 + 0 -> 0
Mem.  : UPDATE     : (tip is cheap) = 0
Mem.  : FIRED      : r2
Mem.  : TRY        : r4
Mem.  : LOOKUP     : (service is goodService) = 1
Mem.  : UPDATE     : (tip is average) = 1
Mem.  : FIRED      : r4
Mem.  : DE-FUZZIFY : centroid @ tip
Mem.  : LOOKUP     : (tip is cheap) = 0
Mem.  : LOOKUP     : (tip is average) = 1
Mem.  : LOOKUP     : (tip is generous) = 0
Mem.  : DE-FUZZIFI : tip = 7.5
Tip = 7.5

?- uncertainty_notrace.
yes

?- findtip(2, 4, Tip).
Tip = 7.5

?-
```

8 Implementação de um sistema de inferência decomposicional

[Esta matéria já não é dada em TSI]

Nesta secção analisa-se a implementação de um sistema de representação e de raciocínio baseado na lógica vaga, em que o conhecimento é representado através de regras de Zadeh-Mamdani e o raciocínio é encadeado para trás e recorre ao método de inferência decomposicional.

Antes de se analisar o programa, apresentam-se algumas decisões referentes à representação de universos de discurso, de conjuntos vagos, de proposições vagas elementares, de proposições vagas compostas através de conectivas e regras vagas. Depois, clarifica-se a natureza das perguntas que se podem efetuar a um sistema deste tipo.

A base de conhecimento de um sistema baseado em lógica vaga tem que incluir a definição de universos de discurso, de conjuntos vagos, de regras e de factos. A Figura 14 exemplifica a representação de conjuntos vagos e dos correspondentes universos de discurso.

```
% Universes of discourse
universe(smoking_universe, [0, 2, 4, 6, 10]).  % Number of cigarretes
universe(hart_attack_risk_universe, [1, 2, 3, 4, 5]). % Risk
% Fuzzy sets
% Smoking
fSet(heavy_smoking, smoking_universe, [0, 0.1, 0.6, 0.8, 1.0]).
fSet(moderate_smoking, smoking_universe, [0, 0.4, 1, 0.4, 0]).
% Hart Attack Risk
fSet(high, hart_attack_risk_universe, [0, 0.2, 0.7, 0.9, 1.0]).
fSet(medium, hart_attack_risk_universe, [0, 0.3, 1, 0.3, 0]).
```

Figura 14 – Representação de conjuntos vagos

Cada conjunto vago é definido num determinado universo de discurso. Usaram-se os predicados *universe/2* para definir universos de discurso e *fset/3* ("fuzzy set") para definir conjuntos vagos. O primeiro argumento de *universe/2* é a designação do universo de discurso. O segundo argumento é o conjunto dos seus valores. O primeiro argumento de *fset/3* é o nome do conjunto vago. O segundo argumento é a designação do universo de discurso para o qual o conjunto vago está definido. Finalmente, o terceiro argumento de *fset/3* é uma lista que representa a sua função de pertença do conjunto.

As proposições vagas elementares (x is A) são representadas através do predicado *f_is/2* ("fuzzy is") já que o predicado *is/2* é reservado para o Prolog.

Uma regra de Zadeh-Mamdani é uma estrutura condição-conclusão em que a condição pode ser uma proposição vaga elementar ou uma conjunção de proposições vagas elementares; e em que a conclusão é uma proposição vaga elementar:

IF (x_1 is A_1) and (x_2 is A_2) and … and (x_n is A_n) THEN (y is B)

Como habitualmente, a representação da conjunção será feita em Prolog através da vírgula. A regra será representada através de um predicado *rule/2* em que o primeiro argumento é a condição e o segundo é a conclusão. Se a condição tiver mais que uma proposição elementar, elas terão de ser envoltas em parêntesis para não se confundirem com diversos arguentos do predicado *rule*. A Figura 15 ilustra a representação de duas regras vagas.

```
% Fuzzy rules
rule(f_is(smoking, heavy_smoking), f_is(hart_attack_risk, high)).
rule(
   f_is(smoking, moderate_smoking),
   f_is(hart_attack_risk, medium)
).
```

Figura 15 – Regras vagas

A primeira regra capta o conhecimento "se a quantidade de tabaco é elevada, o risco de ataque cardíaco é elevado". A segunda regra representa o conhecimento "se a quantidade de tabaco é moderada, o risco de ataque cardíaco é médio".

Os factos de um sistema baseado em lógica vaga são proposições vagas, as quais podem recorrer a conjuntos vagos rotulados existentes no sistema ou a conjuntos vagos não rotulados. Nesta implementação, os factos que recorrem a conjuntos vagos rotulados representam-se através do predicado *f_is/2*; os factos que recorrem a conjuntos vagos não rotulados no sistema, representam-se através do predicado *unlabled_fuzzy_prop/3* que define o universo de discurso e o valor vago de uma determinada variável vaga. O primeiro argumento de *unlabled_fuzzy_prop/3* é o nome da variável vaga. O segundo argumento é o nome do universo de discurso da variável. O terceiro argumento de *unlabled_fuzzy_prop/3* é a função de pertença do valor vago da variável. A Figura 16 mostra um facto representado através de

uma proposição vaga que recorre ao conjunto vago rotulado *"heavy_smoking"*, e um facto que recorre a um conjunto vago não definido no sistema.

```
% Facts

f_is(smoking, heavy_smoking)

unlabled_fuzzy_prop(smoking, smoking_universe, [1, 0, 0, 0, 0]).
```

Figura 16 – Dois factos vagos

Os dois factos representam alternativas para a variável vaga *"smoking"*.

Uma pergunta a um sistema baseado em lógica vaga com regras de Zadeh-Mamdani com encadeamento para trás consiste em saber qual o valor vago de uma dada variável vaga. O predicado *fuzzy/3* permite responder a este tipo de interrogações. O primeiro argumento de *fuzzy/3* é uma variável vaga. O segundo argumento de *fuzzy/3* é o universo de discurso dessa variável. O terceiro argumento do predicado *fuzzy/3* representa a resposta, isto é, representa a função de pertença do valor vago da variável vaga y, para todos os elementos do seu universo de discurso.

```
% Interaction

?- fuzzy(hart_attack_risk, RiskUniverse, Risk).

RiskUniverse = [1, 2, 3, 4, 5]

Risk = [0, 0.1, 0.6, 0.8, 1]
```

Figura 17 – Interrogação a um sistema vago

Face a uma interrogação como a que se apresenta na Figura 17, o sistema tem que encontrar todas as contribuições para o valor para a variável vaga especificada na interrogação.

Para além da representação de conjuntos vagos, de universos de discurso, de factos vagos e de regras vagas, um sistema vago tem que ser configurado. É necessário escolher que implicação vaga usar, qual o método de composição e qual o tipo de ligação (*"else link"*). A Figura 18 ilustra a forma como a configuração de um sistema vago pode ser feita através dos predicados *implication_relation/1*, *composition_method/1*, e *else_link/1*.

```
% Settings

implication_relation(rc_implication).

composition_method(max_min_composition).

else_link(or_link).
```

Figura 18 – Configuração de um sistema vago

Na Figura 18, é esolhida a implicação Rc, a composição Max-Min e a ligação *"Or-link"*.

A explicação do predicado *fuzzy/3* tem duas fases. Primeiro analisa-se a componente da definição de *fuzzy/3* que não dependende de quaisquer escolhas de configuração. Depois, serão analisadas algumas das possibilidades relativas à implicação vaga, à composição e ao *"else link"*.

O predicado *fuzzy/3* recorre ao predicado *contribution/2* para obter todas as contribuições possíveis para o valor da variável da interrogação, determina o universo da variável vaga da interrogação, determina o tipo de *"else-link"* e efetua a combinação das várias contribuições de acordo com o *"else-link"* escolhido.

Supondo que o *"else link"* selecionado é um *"or link"*, a linha *Link =.. [ElseLinkMethod, Univ, L, FVal]* serve para criar o termo *or_link(Univ, L, FVal)*. Esse termo é avaliado no predicado *call/1* para instanciar a variável *FVal* com o resultao da combinação das várias contribuições contidas em *L*.

O predicado *universe_of_var/2* determina o universo de discurso de uma variável vaga. Uma variável vaga aparece em factos ou em regras na base de conhecimentos. Os factos são representados através dos predicados *f_is/2* e *unlabled_fuzzy_prop/3*. As regras são representadas pelo predicado *rule/2*. No caso das regras, basta verificar se a variável aparece na conclusão da regra porque, se aparecer na condição, então aparece igualmente num facto ou na conclusão de outra regra qualquer. O predicado *universe_of_var/2* considera esses três casos.

contribution/2 tem três cláusulas. As duas primeiras cláusulas são para os casos em que existe um facto que estipula o valor da variável da interrogação. A segunda aplica-se ao caso em que o valor da variável é determinado por regras.

```
fuzzy(FVar, Univ, FVal):-
   universe_of_var(Fvar, Univ),
   bagof(Contrib, contribution(FVar, Contrib), L),
   else_link(ElseLinkMethod),
   Link =.. [ElseLinkMethod, Univ, L, FVal],
   call(Link).

contribution(FVar, FVal):-
   unlabled_fuzzy_prop(FVar, _, FVal).
contribution(FVar, FVal):-
   f_is(FVar, SetName),
   fSet(SetName, _, FVal).
contribution(FVar, FVal):-
   rule(Cond, f_is(FVar, B)),
   rule_contrib(Cond, f_is(FVar, B), FVal).

universe_of_var(Fvar, Univ):-
   unlabled_fuzzy_prop(Fvar, UName, _),
   universe(Uname, Univ), !.
universe_of_var(Fvar, Univ):-
   f_is(Fvar, Fval),
   fSet(Fval, Uname, _),
   universe(Uname, Univ), !.
universe_of_var(Fvar, Univ):-
   rule(_, f_is(Fvar, Fval)),
   fSet(Fval, Uname, _),
   universe(Uname, Univ), !.
```

Figura 19 – Predicado *fuzzy/3*

A terceira cláusula de *contribution/2* usa o predicado *rule_contrib/3* para determinar a contribuição de uma regra para o valor de uma variável vaga.

rule_contrib/3 tem dois casos: o caso em que a condição da regra tem uma única proposição atómica, e o caso em que a condição da regra é uma conjunção (Figura 20). Neste último caso, *rule_contrib/3* efetua inferência decomposicional.

```
rule_contrib((Cond1, Cond2), Goal, Contrib):-
   !,
   rule_contrib(Cond1, Goal, C1),
   rule_contrib(Cond2, Goal, C2),
   and_agregate(C1, C2, Contrib).
rule_contrib(f_is(X, A), f_is(Y, B), Contrib):-
   % Implication truth function
   implication_relation(IRelation),
   fSet(A, _, ASet),
   fSet(B, _, BSet),
   Implication =.. [IRelation, ASet,BSet, ISet],
   call(Implication),
   % Composition (MP inference)
   composition_method(CMethod),
   contribution(X, Antecedent),
   Composition =.. [CMethod, Antecedent, ISet, Contrib],
   call(Composition).
```

Figura 20 – Contribuições das várias regras

De acordo com a inferência decomposicional, quando se tem uma regra em que a condição é composta pela conexão de diversas proposições elementares, decompõe-se a regra em várias regras, cada uma delas com apenas uma proposição elementar e todas elas com a mesma conclusão (subsecção 4.2).

De acordo com a inferência decomposicional, uma regra como

IF (x_1 is A_1) and (x_2 is A_2) and … and (x_n is A_n) THEN (y is B)

deve ser decomposta na sequência de regras

IF x_1 is A_1 THEN y is B

IF x_2 is A_2 THEN y is B

…

IF x_n is A_n THEN y is B

A contribuição de cada uma destas regras é determinada pelo método da composição e os resultados são agregados de acordo com a conectiva correspondente na condição da regra de partida (i.e., a regra que foi decomposta).

O método da composição permite determinar a contribuição de cada regra: $B'=A_i' o A_i \rightarrow B$.

O predicado *rule_contrib/3* apenas considera a conjunção de condições porque se assume que temos regras de Zadeh-Mamdani. Se pretendêssemos ter também disjunções, necessitaríamos da seguinte cláusula:

```
rule_contrib((Cond1; Cond2), Goal, Contrib):-
   !,
   rule_contrib(Cond1, Goal, C1),
   rule_contrib(Cond2, Goal, C2),
   or_agregate(C1, C2, Contrib).
```

No caso da contribuição de uma regra com uma única proposição atómica na condição, primeiro, são obtidas as funções de verdade da condição e da conclusão. Depois é determinada a relação de implicação selecionada. A função de verdade da implicação determina-se a partir das funções de verdade do antecedente e do consequente de acordo com a relação de implicação selecionada. Admitindo que a relação de implicação é a implicação Rc, a linha *Implication =.. [IRelation, ASet, BSet, ISet]*, serve para criar o termo *rc_implication(ASet, BSet, ISet)*. Quando este é avaliado pelo predicado *call/1*, *ISet* fica instanciado com a função de verdade da implicação.

A computação, em tempo de execução, da função de verdade de uma implicação é uma opção muito pouco eficiente, especialmente porque cada regra poderá ser usada mais que uma vez. Uma alternativa seria a computação das funções de verdade de todas as regras antes de começar a inferência.

Após a análise da parte independente de escolhas do predicado *fuzzy/3*, passa-se agora à explicação de algumas das alternativas existentes para a implicação, para a composição e para o *"else link"*.

Implicação

A relação de implicação escolhida é usada pelo predicado que determina a função de verdade de cada regra a partir das funçõe de verdade do antecedente e do consequente. A Figura 21 apresenta a definição do predicado *rc_implication/3* o qual determina a função de verdade de uma regra quando a relação de implicação é a implicação Rc.

```
rc_implication(MuASet, [MuB|MuBs], ImplicationTruthFunction):-
   rc(MuASet, MuB, Mu),
   rc_implication(MuASet, MuBs, Mus),
   append(Mu, Mus, ImplicationTruthFunction).
rc_implication(_, [], []).

rc([A|As], B, [Min|Mins]):-
   minimum(A, B, Min),
   rc(As, B, Mins).
rc([], _, []).
```

Figura 21 – Implicação Rc

Considere-se a implicação $P \rightarrow Q$ em que $P \equiv$(x is A), $Q \equiv$(y is B), A é definido no universo U e B é definido no universo V. A implicação define-se recorrendo a uma relação entre os conjuntos vagos A e B, isto é, a função de verdade da implicação aplica cada par (u, v) num valor pertencente ao conjunto [0,1]

em que u e v são elementos de U e de V respetivamente: $\mu_{P\rightarrow Q}$:U×V→[0,1]. Isto significa que é necessário um programa que para cada combinação de u e v, determine o correspondente valor de $\mu_{P\rightarrow Q}$(u, v). O resultado pode visualisar-se através de uma tabela de duas entradas u e v, como se mostra na Figura 22.

	0	2	4	6	v
1	μ(1,0)	μ(1,2)	μ(1,4)	μ(1,6)	
2	μ(2,0)	μ(2,2)	μ(2,4)	μ(2,6)	
3	μ(3,0)	μ(3,2)	μ(3,4)	μ(3,6)	
4	μ(4,0)	μ(4,2)	μ(4,4)	μ(4,6)	

u

Figura 22 – Função de verdade de uma implicação

O programa da Figura 21 representa uma tabela como a da Figura 22, através de uma lista em que os elementos da primeira coluna são seguidos pelos elementos da segunda coluna, e estes são seguidos pelos elementos da terceira coluna, e finalmente pelos elementos da quarta coluna:

[μ(1,0),μ(2,0),μ(3,0),μ(4,0),

μ(1,2),μ(2,2),μ(3,2),μ(4,2),

μ(1,4),μ(2,4),μ(3,4),μ(4,4),

μ(1,6),μ(2,6),μ(3,6),μ(4,6)].

O programa fixa um v e percorre todos os u's. Depois, passa para o próximo v e percorre todos os u's de novo. E continua, até que todos os v's tenham sido percorridos. No caso da implicação Rc, o grau de verdade de (x is A)→(y is B) para um dado par (u,v) é o menor dos graus de verdade de (x is A) e de (y is B).

Além da implicação Rc, existem outras catorze implicações vagas conhecidas, das quais, a tabela da Figura 7 mostra 10. É de todo o interesse que as várias implicações sejam implementadas para que o engenheiro do conhecimento disponha de um leque de escolhas mais amplo.

Composição

Na inferência baseada no Modus Ponens, se tivermos P→Q e P', podemos inferir Q'=P'oP→Q, em que *"o"* é o operador de composição que permite determinar a função de verdade de Q' em termos das funções de verdade de P' e de P→Q. Um dos métodos de composição é o método Max-Min: $\mu_{Q'}(v)=Max\{Min(\mu_{P'}(u), \mu_{P\rightarrow Q}(u, v))\}$. Para cada v, computam-se os mínimos de $\mu_{P'}(u)$ e $\mu_{P\rightarrow Q}(u, v)$ para todos os u's e escolhe-se o máximo desse mínimos.

Em termos visuais (ver Figura 22), fixa-se uma coluna da tabela que representa a implicação e compara-se os valores da função de verdade do antecedente com os valores de verdade dessa coluna, produzindo-se uma nova coluna em que cada elemento é o menor dos dois. Depois, repete-se este procedimento para todas as restantes colunas.

A tabela obtida pelo processo descrito é passada a um predicado que escolhe o máximo dos valores de cada coluna.

```
max_min_composition(Antecedent, [I|Implication], [C|Conclusion]):-
   minima_rest(Antecedent, [I|Implication], Minima, Rest),
   max(Minima, C),
   max_min_composition(Antecedent, Rest, Conclusion).
max_min_composition(_, [], []).

minima_rest([A|As], [I|Is], [M|Ms], Rest):-
   minimum(A, I, M),
   minima_rest(As, Is, Ms, Rest).
minima_rest([], Is, [], Is).
```

Figura 23 – Composição max-min

A tabela apresentada na Figura 22 é representada por uma lista em que os elementos de cada coluna são dispostos em sequência, seguidos dos elementos da coluna seguinte. Esta representação facilita o processo de comparação dos graus de verdade do antecedente com os graus de verdade de cada uma das colunas da tabela.

O predicado *minima_rest/4* compara os elementos de Antecedente com os primeiros N elementos da função de verdade da implicação (i.e., com a primeira coluna da tabela que representa a implicação), produz uma lista com os mínimos dessa comparação, e devolve os restantes elementos da função de verdade da implicação (i.e., as colunas seguintes da tabela).

Por exemplo, *minima_rest([1, 0.5, 0], [0.5, 0.5, 1, 0.3, 0.2, 0, 1, 1, 0.3], Minima, Rest)* instancia *Minima* com a lista [0.5, 0.5, 0], e a variável *Rest* com [0.3, 0.2, 0, 1, 1, 0.3].

O predicado *max/2* determina o maior elemento de uma lista. O predicado *minimum/3* determina o menor de dois valores.

Além da composição Max-Min, também são conhecidas as composições Max-θ e Max-λ (secção 4.2), as quais devem ser implementadas para permitir maior flexibilidade ao engenheiro do conhecimento.

"Else Link"

O predicado que implementa o *"else link"* recebe uma lista de contribuições para o valor de uma variável, geradas por diversas regras, e combina-as para produzir um único resultado. Um dos possíveis *"else links"* é o *"and link"*. Usando o *"and link"*, o resultado será formado, para cada v pertencente a V, pelo menor dos v's das várias contribuições.

```
and_link(_, [X, Y | Rest], Result):-
   and_link(_, [Y|Rest], Tmp),
   and(X, Tmp, Result).
and_link(_, [X], X).

and([X|Xs], [Y|Ys], [Min|Mins]):-
   minimum(X, Y, Min),
   and(Xs, Ys, Mins))).
and([], [], []).
```

Figura 24 – *"And link"*

Dado que o predicado que implementa o "else link" é sempre chamado com três argumentos: universo de discurso da variável vaga, lista de contribuições para essa variável, e resultado, o predicado *and_link/3* também recebe esses três argumentos. No entanto, o and link não necessita de usar o universo de discurso.

O *"or link"* implementa-se exatamente da mesma forma, trocando o mínimo pelo máximo.

A ligação de qualificação da verdade (*"truth qualification link"*) é um pouco mais complicado. É necessário determinar o Ti associado a cada regra e escolher a contribuição da regra com o maior Ti.

A ligação aditiva (*"additive link"*) efetua uma média ponderada dos valores de verdade para cada v pertencente a V. O peso de cada regra poderá ser o Ti dessa regra, obtido como na ligação de qualificação da verdade.