

Documentação técnica (2ª fase)

- secção destinada à função merge-rede -

- para ser considerada em conjunto com a documentação técnica desenvolvida na 1ª fase -

A função merge-rede foi implementada numa forma bastante simples e reutilizadora de código desenvolvido na 1ª fase da rede semântica.

A função garante a actualização dos identificadores dos nós asserção ou hipótese e a não introdução de conceitos já existentes na rede em memória.

Assim a função pode-se considerar completa em todos os sentidos, na medida em que funciona numa forma transparente ao ponto de equivaler a uma interacção directa dum utilizador com a rede.

Este documento pretende descrever o algoritmo utilizado em merge-rede, em linguagem natural. Descrevem-se ainda todas as funções presentes no ficheiro merge.lsp, o qual contém todo o código necessário para a implementação das possibilidades de merge.

Esta documentação não deve no entanto considerar-se isoladamente da respectiva documentação interna, a qual contém comentários relevantes para compreender o funcionamento isolado de cada função desenvolvida para o efeito.

Algoritmo de funcionamento da função merge-rede:

1) Reutilizar o código da função carrega (desenvolvida na 1ª fase da rede semântica) para construir uma função muitíssimo semelhante, mas que faça o carregar da rede contida num ficheiro; não para a variável global *rede*, mas sim para uma nova variável global chamada *buffer*.

2) A função que o ponto 1 refere acabou por chamar-se carrega-buffer.

Assim, após a actuação de carrega--buffer, dispomos em *buffer* da rede existente no ficheiro que se quer fazer merge.

3) Vamos consultar cada um dos nós presentes na rede *buffer* e, para os que forem do tipo asserção ou hipótese, vamos determinar a chamada que deveria ser feita a build ou asserção para os criar!

Faz-se então a chamada devida. A chamada feita, encarrega-se de verificar todos os restantes pormenores, como a existência dum conceito equivalente ou dum nó com o mesmo nome. A forma como o princípio da unicidade (verificação dum conceito igual) e o manter dos identificadores dos nós é feita, é completamente deixada a cargo das funções build e asserção!

Como tal, merge-rede não representa um acréscimo de mecanismos de verificação, na medida em que todos os anteriormente disponíveis são reutilizados.

A ter em conta: O ficheiro Merge.Lsp contém todo o código necessário para a implementação da função merge-rede. Nesse ficheiro existem funções claramente obtidas de outras já existentes no ficheiro carrega.lsp; variando apenas na variável para a qual trabalham (*rede* ou *buffer*).

É óbvio que; para evitar a repetição de código que daqui advém, poder-se-ia ter recorrido à passagem de parâmetros para uma função que depois decidiria se estava a trabalhar para *rede* ou para *buffer*, no entanto; e porque essa repetição só ocorre em duas pequenas funções, considerou-se ser mais vantajoso individualizar o código da possibilidade de merge do código do resto do programa, de forma a atingir-se o que se atingiu no ficheiro merge.lsp; isto é, reunir nele todas as funções precisas para merge-rede.

Funções presentes em Merge.Lsp (e respectiva documentação técnica):

1. Carrega-Buffer (nome-dum-ficheiro)

Esta função procede à leitura do ficheiro argumento, tal qual a função carrega faria. As diferenças relativamente a carrega são que a leitura das relações é agora, naturalmente, feita para cima das relações já existentes não sendo mantidos; no entanto, nomes de relações repetidos.

Esta função invoca as funções carrega-nós-buffer e carrega-relações-buffer da mesma forma que a função carrega fazia com as funções carrega-nós e carrega-relações. Isto quer dizer que, primeiro criam-se os nós requisitados no ficheiro, com as relações ascendentes e descendentes a nil. A actualização das relações será depois feita por carrega-relações-buffer.

Esta função acaba por provocar o carregar duma rede semântica na variável global *buffer*, tal qual a função carrega provocaria para a variável *rede*.

2. Carrega-nós-buffer (lista)

Recebe uma lista da forma ((nome-1 tipo-1)... (nome-n tipo-n)), e procede ao invocar da função cria-no para criar os nós com identificadores até nome-n, e com tipo igual a tipo-n. Nesta altura não são conhecidas as relações dos nós, pelo que a invocação de cria-nó, é feita com o campo das relações descendentes e das relações ascendentes a nil.

3. Obtém-relações (lista)

Recebe uma lista de relações descendentes para certo nó do tipo hipótese ou asserção. Determina então que chamada deveria ser feita a build ou asserção (respectivamente) para criar um nó com aquela lista de relações descendentes.

4. Merge-rede (nome-dum-ficheiro)

O algoritmo para esta função está explicado mais acima.

Esta função utiliza todas as anteriores e é a directamente responsável pela possibilidade de merge. É a única função presente neste ficheiro, que se destina ao utilizador.

5. Existe-nó-buffer (identificador &optional tipo)

Devolve o nó com o identificador (ou com o identificador e tipo) requisitado(s), caso ele exista em *buffer*. Caso contrário devolve nil.

A sua actuação é exactamente igual à da função existe-nó, excepto no que se refere à variável onde a pesquisa é feita. Esta repetição de código está justificada mais acima, neste mesmo documento.

De todas as funções atrás descritas, só merge-rede se destina ao utilizador da rede semântica; todas as outras existem em função de merge-rede e são como ferramentas de nível de inferior necessárias para a possibilidade de merge.

Esta documentação não deve ser tida em conta individualmente; é antes para ser considerada em conjunto com todas as outras documentações técnicas e do utilizador, desenvolvidas quer para esta 2ª fase, quer para a 1ª fase.