

# Documentação técnica (2ª fase)

- secção destinada às funções de procura -

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

Para implementar, tal como requisitado, a procura dum nó na rede, foram criados 2 novos ficheiros - PROCURA.LSP e VAR.LSP - os quais contêm todas as funções necessárias à implementação referida.

Mais uma vez, a fonte inspiradora para o desenvolvimento das possibilidades de procura, foi o SNePS. É claro que a rede semântica em desenvolvimento não desempenha as suas várias tarefas (a de procura incluída) como o SNePS faz; em muitas situações, o pequeno conjunto de funções permitidas ao utilizador, têm uma interface distinta em pontos tão sensíveis como as variáveis.

O objectivo deste documento - que deve ser considerado em conjunto com o código fonte e respectivos comentários - é tão somente explicitar as ideias que estiveram por detrás do desenvolvimento do código. A forma de usufruir das novas funções deverá ser consultada na nova documentação para o utilizador.

Assim, o desenvolvimento das funções de procura começou com a ideia de que seriam necessárias fazer duas funções: procura e procura-faz; que corresponderiam de alguma forma, às funções find e findorbuild (respectivamente) oferecidas pelo SNePS.

Pretendia-se que essas funções tivessem uma sintaxe de chamada com uma versatilidade o mais semelhante possível à do SNePS, apesar das diferenças óbvias no «aspecto» da chamada, nomeadamente no facto de não se seguir a convenção «unquoted». Acima de tudo, acabou-se por desenvolver uma forma de suporte a variáveis bastante acessível e com exigências a nível de software não muito elevadas o que permite uma fácil manutenção das funções envolvidas.

Conforme será possível constatar pela listagem (anexa) com o código fonte para as possibilidades de procura (PROCURA.LSP) e de suporte a variáveis (VAR.LSP), desenvolveram-se as seguintes funções:

## **Ficheiro PROCURA.LSP:**

### **1.**

#### **forma-lista-nomes (lista-nos)**

- função que recebe uma lista cujos elementos são estruturas representativas de nós (consultar documentação da 1ª fase para averiguar como é a estrutura nó) e que devolve uma lista cujos elementos são os identificadores de cada um dos nós presentes na lista de nós argumento.

## 2.

### **procura (&rest argumentos)**

- será esta a nova função mais invocada pelo utilizador da rede.  
a sintaxe respectiva deverá ser:

(procura 'relação1 'conjunto-de-nós1... 'relação-n 'conjunto-de-nós-n)

A filosofia por detrás desta chamada é:  
pretende-se ver devolvida a lista cujos elementos são os identificadores dos nós que verificam as condições expressas na chamada. Neste caso, as condições expressas na chamada são:

- quero obter o(s) nó(s) que têm uma arco chamado relação1 para cada um dos nós cujos identificadores estão contidos em conjunto-de-nós1,....., e o arco relação-n para cada um dos nós cujos identificador esteja contido em conjunto-de-nós-n.

É preciso ter em conta que os conjunto-de-nós podem ser conjuntos singulares, ou não, e que podem ser mencionados explicitamente (isto é, o utilizador escrever o identificador de cada um dos nós desejados) ou através da função valor? conforme deverá ser consultado na documentação para o utilizador!

Em linguagem natural, a abordagem para esta função foi a seguinte:

- achar os nós que verificam as condições expressas.  
- criar (se necessário) e atribuir (se existirem) os valores devidos às variáveis eventualmente requisitadas no chamada da função procura; deverá consultar a documentação do utilizador para saber como requisitar variáveis.

O algoritmo pensado foi o seguinte:

**a)** pôr a lista de argumentos (o que o utilizador escreveu) numa forma mais fácil de manipular pelo programa. Isto é, se o utilizador invocar, por ex., (procura 'membro '(a b c)), a lista de argumentos deverá ser transformada no formato ((membro a) (membro b) (membro c)).

A função que ficou encarregue de dar este aspecto à lista de argumentos é a «trata-argtos» (referida à frente).

**b)** invocar uma função (que acabou por se chamar procura2) que deverá achar os nós que verificam as condições expressas. Esta função é descrita mais à frente. Por uma questão de abstraccionismo, considere-se por agora que ela receberá os argumentos devidamente tratados e devolverá uma lista cujos elementos são os próprios nós (e não somente os seus identificadores) que verificam as condições estipuladas.

c) passar à fase de eventual criação e atribuição de valores a variáveis. Volta a ser necessário dar um formato mais prático à lista de argumentos, desta feita isso corresponde a formar duas listas, ambas do formato ((relação nó)... (relação nó)) que referem os pares (relação nó) que já não vale a pena analisar, pelo facto de não referirem a necessidade de criação de variáveis; e os pares (relação nó) que exigem posterior tratamento por requererem a criação de variáveis!

Estes pares atrás referidos, são tais que o nó mencionado, não é mais do que o nome da variável cuja criação se requisita e que deverá vir a conter como valor os identificadores dos nós apontados por cada um dos nós solução devolvidos em b) através do arco identificado por «relação».

**A saber:** a formação das duas listas de pares (relação nó) - uma para os nós excluídos de posterior análise e outra para os que requisitam variáveis - fica a cargo da função «forma-excluídos-e-vars» (referida à frente).

d) uma vez conhecidas as variáveis a criar, são criadas aquelas que ainda não existirem e, em qualquer caso, actualizados os respectivos valores.

A atribuição dos valores devidos às variáveis devidas, fica a cargo da função «forma-var» (referida mais à frente).

e) resta invocar a função «forma-lista-nomes» (já referida) para devolver a lista dos identificadores dos nós que verificaram as condições de procura.

### 3.

#### **proc (argumentos)**

Esta função faz aquilo que se disse para a função procura.

Na realidade, a função procura não faz mais que chamar esta função proc. Optou-se por um mecanismo deste género de forma a permitir que a função «procura-faz» (referida mais à frente) pode-se actuar tal qual a função «procura», no caso da existência do nó (consultar a documentação do utilizador para saber o que é suposto a função «procura-faz» fazer).

#### 4.

##### **trata-argtos (lista-argumentos)**

Tal como referido a propósito da função «procura», limita-se a receber uma lista de argumentos - que correspondem à forma de invocação da função «procura» e devolver uma outra lista que torna mais prática a manipulação requisitada pelo utilizador. Assim, se por exemplo receber (membro (a b c)), vai devolver ((membro a) (membro b) (membro c)).

Esta função é no entanto utilizada ainda para analisar a correcção sintáctica da chamada do utilizador; isto é, avisar quando se escreve o nome duma relação que ainda não foi definida, quando faltam argumentos ou quando se esperavam nós e afinal não surge nada.

#### 5.

##### **relação? (identificador-duma-relação)**

Devolve nil se o identificador recebido não corresponder a nenhuma das relações já definidas pelo utilizador. Devolve t caso contrário.

#### 6.

##### **junta-par (lista relação nós)**

Invocada pela função «trata-argtos».

Recebe o nome duma relação (relação) e uma lista cujos elementos deverão ser identificadores de nós (nós); deverá devolver a lista argumento (lista) com os pares (relação x) acrescentados no final, sendo x, consecutivamente, cada um dos elementos da lista nós.

Se por exemplo receber relação=membro e nós=(a b), deverá devolver a lista argumentos com os elemento (membro a) e (membro b) acrescentados no final.

7.

### **procura2 (lista-de-nós-onde-procurar argumentos)**

Muitíssimo importante esta função. É ela que, conforme referido a propósito da função «procura», faz o verdadeiro trabalho de devolução dos nós que cumprem as condições que o utilizador especificou para a procura.

Deverá ser inicialmente invocada com `lista-de-nós-onde-procurar=*rede*` (ver `*rede*` na documentação técnica da 1ª fase) e com `argumentos=lista` de argumentos já processados, isto é, na forma `((membro a) (membro b)...) , por ex.`

Para cada um dos pares (relação nó) contidos na lista de argumentos, vai chamar a função «selecciona» a qual deverá ir actualizando progressivamente a lista a devolver por «procura2».

A lista a devolver começou por ser a própria `*rede*`, mas a função `selecciona`, ir-se-á encarregar de reduzir a devolução aos nós que verificam as condições; o funcionamento de «selecciona» é descrito já a seguir.

8.

### **selecciona (lista-de-nós-onde-seleccionar par-relação-nó)**

A `lista-de-nós-onde-seleccionar`, começa por ser `*rede*`, conforme referido atrás. O `par-relação-nó`, deverá ser da forma (relação nó).

Se o nó no `par-relação-nó` for um identificador para uma variável, então os nós da `lista-de-nós-onde-seleccionar` que tenham a relação cujo identificador é o primeiro elemento do `par-relação-nó`, são seleccionados para serem devolvidos.

Se o nó no `par-relação-nó` não for um identificador dum variável (condição essa determinada pela função «var?») mas sim um identificador dum nó mesmo, então serão seleccionados para devolução os nós da `lista-de-nós-onde-seleccionar` que tenham a relação descendente mencionada para o nó referido.

Porque a `lista-de-nós-onde-seleccionar` vai sendo progressivamente menor, e será sempre completamente analisada, progressivas selecções nessa lista asseguram que no final sejam devolvidos os nós que verificam TODAS as condições que o utilizador especificou atrás na chamada a «procura».

Para saber quais os nós que têm certa relação descendente a sair deles ou quais os nós que têm certa relação descendente para certo nó em particular, existem as funções «tem-rel?» e «tem-rel-para?», a seguir descritas.

**9.**

**tem-rel (nó relação)**

Devolve t o nó em questão tiver a relação descendente "relação".  
Devolve nil caso contrário.

**10.**

**tem-rel-para (nó relação identificador-do-nó-destino)**

Devolve t se o nó em questão tiver a relação descendente "relação" para o nó cujo identificador é identificador-do-nó-destino.  
Devolve nil caso contrário.

**11.**

**tira-primeiro (identificador)**

Devolve um identificador idêntico ao identificador recebido, mas com o primeiro carácter retirado.

Se o identificador recebido consistisse só em ? devolveria -?.

É uma função utilizada para atribuir nomes às variáveis e necessária pelo facto de uma requisição de criação de variável (ou actualização do seu valor) ser possível, só fazendo anteceder o nome da variável por ?.

**12.**

**forma-excluídos-e-vars (argumento)**

Recebe uma lista que corresponde à lista de argumentos (já processados pela função «trata-argtos» e devolve uma lista que consiste em 2 listas; a primeira delas é a lista dos pares (relação nó-id) que já estão excluídos de poderem ser utilizados para a criação (ou actualização de variáveis) devido ao facto de nó-id não ser um identificador destinado a uma variável. A segunda lista contém os pares (relação nó-id) em que nó-id é, na realidade o identificador duma variável.

**13.**

**arranja-nomes (lista-de-relações-descendentes)**

Recebe uma lista cujos elementos são estruturas relação (consultar a documentação técnica da 1ª fase) e devolve uma lista da forma ((relação nó-id)... (relação nó-id)) em que relação é um identificador para a relação cuja estrutura pertence à lista argumento e nó-id é o identificador do nó apontado por essa relação (descendente).

**14.**

**forma-var (relação, nós-devolvidos-pela-procura, nós-destino excluídos)**

Nós-devolvidos-pela-procura, são os nós cujos identificadores vão acabar por constituir a devolução da função de procura. São esses nós que contêm de certeza as instanciações correctas para as variáveis eventualmente requisitadas.

Nós-destino-excluídos: é uma lista cujos elementos são da forma (relação identificador-do-nó-destino), e que correspondem aos pares da mesma forma que o utilizador mencionou aquando da chamada à função procura; pelo facto de já serem conhecidos pelo utilizador, os nós referidos estão excluídos de - para a relação em causa - serem soluções possíveis para o conjunto de valores a (eventualmente) atribuir a certa variável.

Esta função vai acabar por devolver a lista cujos elementos são os identificadores dos nós possíveis de atribuir à variável requisitada pelo utilizador. Ou seja forma o valor da variável!

Consultar a documentação do utilizador para saber como invocar a criação ou actualização de variáveis.

**15.**

**procura-faz (&rest argumentos)**

Esta função têm uma sintaxe de chamada idêntica à da função procura!

Assim (tal como a função procura) invoca a função proc e caso seja informada que não existem nós com as características especificadas nos argumentos, tenta criar um nó com essas mesmas características.

A criação do nó só será possível se não tiverem sido requisitadas variáveis na chamada! Quando essa criação fôr possível, o nó é criado tal qual o seria por uma chamada directa à função build; ou seja, o nó (eventualmente) criado é do tipo hipótese.

Para saber se a chamada inclui a requisição de variáveis, recorre à função «sem-vars», referida a seguir.

## 16.

### **sem-vars (lista-argumento)**

Recebe uma lista-argumento da forma ((relação nó-id)... (relação nó-id)) e averigua se algum dos nó-id corresponde a um identificador reservado para variáveis, isto é, a um identificador começado por ?.

Se isso acontecer, considera que se está a requisitar a criação duma variável e devolve nil para assinalar que existem variáveis.

Caso contrário devolve t - isto é, não existem variáveis.

*Todas as funções atrás descritas, estão no ficheiro PROCURA.LSP, devidamente comentadas. De todas estas funções, só duas destinam-se ao utilizador da rede semântica : Procura e Procura-faz; as restantes funções são serviços de nível inferior utilizados para implementar estas últimas.*

*Esta documentação técnica refere-se unicamente ao ficheiro PROCURA.LSP e só deve ser considerada em conjunto com esse ficheiro e com a documentação técnica desenvolvida para a 1ª fase da rede semântica. Para compreensão da implementação das variáveis, deve também ser complementada com a documentação técnica a elas destinada. Recomenda-se igualmente a leitura da documentação destinada ao utilizador (1ª e 2ª fases) para melhor inserção no contexto aqui assumido..*