

Projecto duma rede semântica¹

Introdução :

Todas as redes semânticas baseam-se no facto do conhecimento ser representado por um grafo rotulado e dirigido, composto por nós e por arcos. Na prática, os conceitos são representados por nós, enquanto que os arcos encarregam-se de associar esse conceitos.

As redes semânticas são métodos de representação de conhecimento cuja principal vantagem é a de poderem explicitar as associações entre conceitos.

A seguinte documentação visa esclarecer sobre as possibilidades dum projecto duma rede semântica, projecto esse ainda numa primeira fase do seu desenvolvimento. Pretende-se esclarecer o eventual utilizador da rede sobre as estruturas disponíveis e ferramentas existentes para a sua manipulação.

Utilizadores mais interessados e com um mínimo conhecimento da linguagem Common Lisp, encontram no final da documentação os pormenores relacionados com a implementação dos vários serviços já oferecidos nesta fase precoce de desenvolvimento da rede.

Para o realmente interessado no programa, faz-se acompanhar uma listagem de todo o código fonte desenvolvido nesta fase.

Exceptuando a parte final deste documento, a listagem acima referida consiste na documentação (interna) técnica; isso porque, na medida do sensato, é uma listagem em que cada função está documentada em relação aos seus argumentos e devoluções e ainda em relação a um ou outro ponto relevante do seu código.

¹ Nota: A documentação que se segue refere-se apenas à 1ª fase de desenvolvimento da rede semântica. Durante a leitura, deverá estar sempre presente o facto de também se enviar a 2ª e derradeira fase do projecto, juntamente com a documentação respectiva.



Documentação para o utilizador

A rede a que esta documentação se refere, está a ser desenvolvida tendo por base o SNePS - Semantic Network Processing System. Assim é de esperar uma semelhança em relação a certo tipo de possibilidades da rede com possibilidades do SNePS.

Na rede que se desenvolve, respeita-se o princípio da unicidade, isto é, cada nó representa um e um só conceito; ao passo que cada conceito, é representado por um e um só nó.

De entre as funções já desempenháveis pela rede nesta fase do seu desenvolvimento, existem as relacionadas com a definição de relações, com a adição de conhecimento e inspecção do conhecimento. Já é entretanto possível remover conhecimento, bem como arquivar e recuperar em / de ficheiros o conhecimento que a rede pretende representar.

De entre as várias funções implementadas, só algumas se destinam ao utilizador da rede. Muitas das funções existem por necessidade de outras funções de nível progressivamente inferior, podendo-se considerar que o utilizador trabalha com as funções ou macros de nível superior.

São precisamente as funções destinadas ao utilizador que se descrevem nas próximas páginas. Tendo em conta o alto nível dos serviços referidos, opta-se por só fazer uma descrição dos efeitos desses mesmos serviços sem atender a questões de pormenores de implementação ou funcionamento.

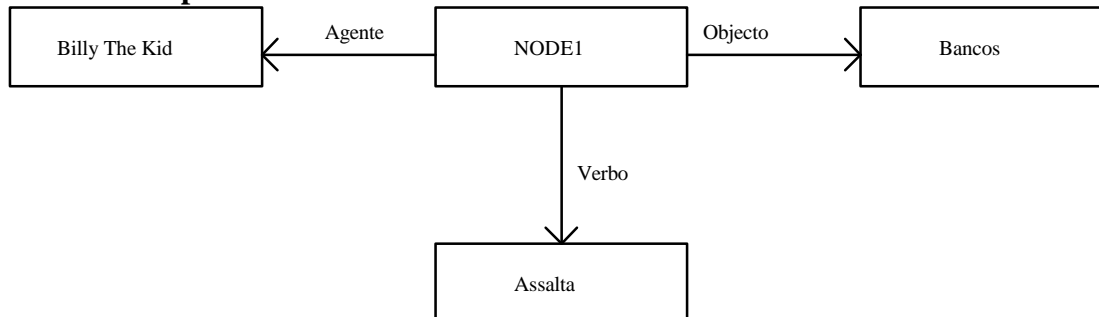
Como todas as redes, a rede implementada (fase 1 da implementação) consiste num grafo rotulado e dirigido. Um grafo tem nós e arcos. Os arcos tem um sentido no caso dos grafos dirigidos. Os nós vão corresponder aos conceitos e os arcos às relações entre esses conceitos. No caso duma rede baseada no SNePS, pode-se ainda dizer que muitos dos nós são representativos de proposições.

Na próxima página estão alguns exemplos de proposições representadas por grafos e suportadas na rede aqui discutida, já nesta fase da sua implementação. A forma de gerar conceitos (nós) que representem as proposições seguintes será discutida também, um pouco mais à frente.

Assim, os seguintes grafos representam proposições e correspondem a conhecimentos possíveis de capturar em SNePS e na rede objecto de discussão.



Exemplo 1:



Neste grafo, temos 4 nós: NODE1, Billy-the-Kid, assalta e banco. Temos ainda 3 relações: Agente, Verbo e Objecto. De todos os nós presentes, o que «reúne mais informação» é o nó NODE1, na medida em que se refere a todos os outros. NODE1 refere que Billy the Kid é o agente do verbo assalta em relação ao objecto banco.

Por outras palavras, NODE1 está a representar a proposição: "Billy The Kid Assalta Bancos."

Como se verá a seguir, uma única linha, representando um único comando, seria capaz de capturar este conhecimento na rede.

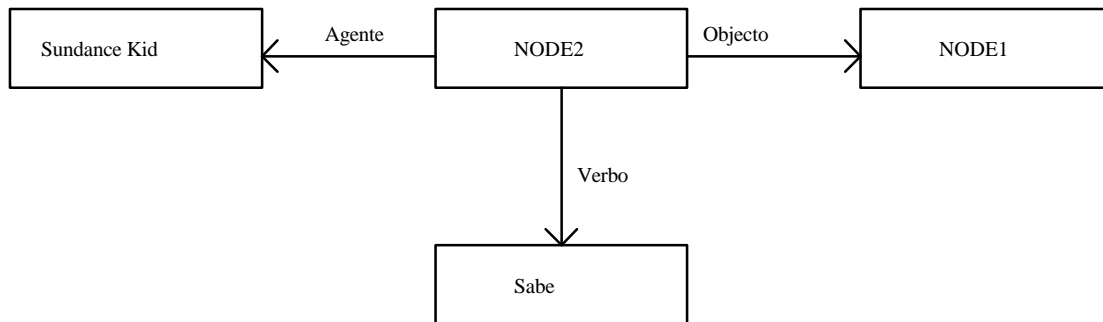
Conforme o leitor suspeitará, NODE1 não é um nó como os outros. NODE1 é um nó que a rede automaticamente gera quando instruída para representar determinado conhecimento proposicional. No entanto NODE1 pode ser utilizado por outros nós que queiram representar conhecimento um pouco mais elaborado.

Por exemplo, suponha-se que queria-se dizer que Sundance Kid sabe que Billy the Kid assalta bancos. Ocorre naturalmente a ideia de aproveitar NODE1 para a construção do grafo que represente esta informação. Aliás, o princípio da unicidade, deverá garantir que a informação relativa a Billy the Kid não se repete!

Ver a próxima página para saber qual a representação correcta em grafo deste problema.

Exemplo 2:





Assume-se que NODE1 é tal como está definido na página anterior. Desta forma pode-se considerar que NODE2 está a representar a informação "Sundance Kid sabe que Billy the Kid assalta bancos."

Este segundo exemplo também é capturável na rede semântica através duma única instrução, isso assumindo que o NODE1 já tinha ele próprio sido criado por uma instrução semelhante. As relações presentes neste novo grafo são as mesmas que as existentes no grafo anterior. Surgem agora mais 3 nós - o NODE2, Sundance Kid e Sabe; o NODE1 já existia no grafo 1 e é como que reutilizado aqui no grafo 2.

Estes 2 exemplos deixam antever essencialmente dois tipos de nós.

Na realidade podem estar presentes até 3 tipos de nós nestes exemplos. E a rede está preparada para suportar 4 tipos de nós. Explica-se:

Os nós como Sundance Kid, Billy the Kid, Sabe, Assalta e Bancos são nós ditos constantes na medida em que são conceitos que não variam, também chamados de não estruturados.

NODE1 e NODE2 podem ser - para a rede em implementação - de dois tipos: podem ser assercoes ou hipóteses. O que os distingue é acima de tudo a rede acreditar que os nós asserção são verdades absolutas, enquanto que os nós hipótese, são só isso - hipóteses.

O tipo de NODE1 e NODE2 tem influência em operações como a eliminação de conceitos da rede, conforme será referido aquando da discussão das operações de remoção de conhecimento permitidas ao utilizador.

O outro tipo possível para nós é o tipo variável. Nesta fase da sua implementação, a rede permite a existência de nós variáveis embora os trate de forma idêntica a todos os outros. Os nós variáveis são por enquanto como os restantes exceptuando que se assinala o seu tipo como VAR e que os seus nomes devem começar por \$.

Para o utilizador interessará acima de tudo distinguir por enquanto entre nós constantes, nós hipótese e nós asserção.



Também as relações - ou arcos - podem ser de dois tipos. Existem as relações descendentes, como agente, verbo e objecto nos exemplos anteriores; e as respectivas relações inversas: agente-, verbo- e objecto- que a rede implementa automaticamente sem que o utilizador tenha necessidade de definir aquando da criação de arcos.

E pronto, o utilizador é suposto já conhecer a existência de 4 tipos de conceitos/nós e 2 tipos de relações/arcos. É chegada a altura de saber como criar relações, nós, apagar informação, ver descrita informação sobre nós, etc.

O programa que suporta a rede foi escrito em Common Lisp e corre em principio em todas as máquinas com versões de Lisp razoavelmente próximas do Common Lisp. Em particular, o código fonte foi desenvolvido em ambiente VAX/VMS utilizando Common Lisp instalado nessa máquina.

Uma vez na área adequada, para correr o programa bastará teclar LISP para aceder ao Common Lisp. Na prompt Lisp> escrever (load 'run).

As várias funções do programa são carregadas em memória e a partir desse momento, o utilizador fica apto a entrar qualquer dos comandos descritos nas páginas seguintes.

Aviso: Muitas funções para lá das a seguir descritas podem ser invocadas na prompt do Lisp provocando eventualmente efeitos que afectam o comportamento previsto da rede!

O utilizador não deverá manipular directamente as variáveis *rede*, *relacoes* e *num-nos*! Se o fizer é muito provável que descontrolo a actuação das funções a ele (utilizador) especialmente destinadas.

As funções que o utilizador pode mas NÃO DEVE invocar são todas as carregadas em memória mas não documentadas a seguir. O utilizador deve restringir a sua actuação às funções e macros a seguir descritas.

Comandos destinados ao utilizador

1. Definição de relações.

Estas operações permitem introduzir relações estruturais na rede semântica. Só com a existência de relações é possível associar os conceitos isolados de forma a fazer existir um conceito estruturado.

A definição de arcos ascendentes e descendentes é feita pela função def-rel :

Após execução da função (def-rel 'r1 'r2... 'rn) ficam definidas as relações descendentes r1,r2,..., rn e (implicitamente) as relações ascendentes r1-,r2-,..., rn-.



A relação ri é a inversa de ri- e vice-versa.

Exemplo: Para definir as relações descendentes agente, verbo e objecto utilizadas nos exemplos 1 e 2, dever-se-ia fazer:

```
(def-rel 'agente 'verbo 'objecto)
```

Também ficariam definidas as relações ascendentes agente-, verbo- e objecto-. Ter em conta que - internamente - as relações descendentes vão sendo progressivamente acrescentadas ao fim da lista representada pela variável global *relacoes* de forma a que o último elemento dessa lista seja sempre a última relação criada.

As relações ascendentes não ficam arquivadas em nenhuma lista, mas a sua existência é tida como implícita e é reflectida nas relações ascendentes entre arcos.

Ao escrever *relacoes* na prompt do Lisp ficará a saber todas as relações que já definiu desde o início da interacção com a rede.

2. Adição de conhecimento

A partir do momento em que tenhamos relações estruturais podemos introduzir conceitos na rede. A introdução de nós isolados na rede não é possível na medida em que, sózinho, um nó não pode representar um conceito.

Um conceito só tem significado pelas relações existentes com outros conceitos. Um nó de base ou constante só poderá existir em conjugação com outros nós.

Existem duas funções para adição de conhecimento na rede. Para compreensão destas funções, o utilizador deverá saber que, de alguma forma, internamente, é guardado o tipo de cada nó e que a diferença entre as duas funções a seguir descritas é só relativa ao tipo do nó criado; assim:

```
(build 'r1 'conjnós1 'r2 'conjnós2... 'rn 'conjnós)
```

ou

```
(assercao 'r1 'conjnós1 'r2 'conjnós2... 'rn 'conjnós)
```

Em que:

r1... rn - são relações previamente definidas, isto é, devem já ter sido criadas por def-rel.

Caso alguma das funções não tenha sido criada, build aborta a operação com uma mensagem alusiva à existência duma relação ainda não criada.

conjnós1... conjnós n - podem ser duas coisas.

Podem ser nós isolados, isto é pode-se por exemplo escrever (build 'a 'b) em que b representa um único nó.



No entanto pode-se fazer uma chamada da forma
(build 'relacao '(no1 no2... non))
que é equivalente a uma chamada da forma
(build 'relacao 'no1 'relacao 'no2... 'relacao 'non)).

Por enquanto estas são as únicas formas sintacticamente correctas de chamar build.

Qual o resultado duma destas chamadas acertadas ?

- Tendo em conta o número de nós que desde o início foram criados por build ou assercao (mesmo que alguns tenham depois sido removidos), a rede gera automaticamente um identificador para o novo nó criado como resultado desta chamada. O nome por defeito é NODEn em que n é um inteiro que será 1 aquando do primeiro build ou assercao e que se vai incrementando à medida que se fazem mais chamadas do género.

O novo nó com identificador NODEn fica com as relações descendentes r_1, \dots, r_n para cada um dos nós (eventualmente só 1) presentes em $conjnós_1, \dots, conjnós_n$.

- Se o nó for criado por build, o seu tipo fica Hipótese - o que quer dizer que a rede não acredita necessariamente no nó.

- Se o nó for criado por assercao, o seu tipo fica asserção.

Ter em conta que os nós em $conjnós_n$ que ainda não existam são criados. Caso já existam, são actualizadas as respectivas relações ascendentes e descendentes, caso necessário.

Internamente o tipo hipótese é representado por 'Hip, enquanto o tipo assercao é representado por 'true. Esta informação não é no entanto minimamente relevante para o utilizador da rede, deixando-se apenas a título de curiosidade.

Tanto build como assercao fazem verificação da sintaxe de entrada e enviam ao utilizador mensagens de erro adequadas ao eventual erro cometido.

Cada um dos nós presentes em $conjnós_n$ pode ser de qualquer tipo, incluindo hipótese ou asserção. Assim o novo nó pode-se referir a nós criados anteriormente de forma semelhante.

Aviso: Apesar de em $conjnós_n$ poderem surgir quaisquer tipo de nós, o utilizador tem de obedecer às seguintes regras:

- o nome dum nó variável deve começar por \$, caso contrário será considerado que se está a referir a um nó constante ou hipótese ou asserção.

7



Por exemplo, dever-se-á escrever (build 'r '\$ola) se se quiser que o nó referido seja do tipo variável.

- os nós hipótese ou asserção referidos devem já existir!

Caso o utilizador escreva por exemplo (build 'r 'node2) e NODE2 ainda não tenha ele próprio sido criado por um build ou assercao, o seu identificador passa a ser NOD2 de forma a não confundir com o identificador automaticamente gerado para nós hipótese ou asserção.

E ainda: Conforme já foi referido, nesta primeira fase de desenvolvimento da rede semântica, os nós variáveis estão a receber um tratamento igual ao dos outros nós, exceptuando na sintaxe a que o seu identificador deverá obedecer e no assinalar do seu tipo como var.

Exemplo de build e assercao:

- para criar o NODE1 do exemplo 1 deverá escrever:

(build 'agente 'Billy 'verbo 'assalta 'objecto 'bancos)

ou

(assercao 'agente 'Billy 'verbo 'assalta 'objecto 'bancos)

conforme queira que NODE1 seja - respectivamente - uma hipótese ou uma asserção.

3. Inspeção do conhecimento.

As funções que se seguem, servem para inspeccionar o conhecimento declarado na rede.

(dump 'no-id)

Em que no-id é o identificador dum nó.

O resultado desta chamada é a descrição do nó cujo identificador é recebido como argumento.

Como é feita essa descrição:

- Surgem todas as informações possíveis em relação ao nó:

Identificador: identificador do nó

Tipo: Tipo do nó

*Relações ascendentes / nó destino:
rel-asc? / no-dest?*



...
rel-asc? / no-dest?

Relações descendentes / nó destino:
rel-desc? / no-dest?

...
rel-asc? / no-dest?

Para nós não existentes surge uma mensagem de erro que informa que o nó em questão não existe.

Por exemplo para o NODE1 gerado por um build, referido no exemplo 1:

(dump 'NODE1)

Identificador: NODE1

Tipo: Hip

Relações ascendentes / nó destino:
Não há relações ascendentes.

Relações descendentes / nó destino:
AGENTE / BILLY-THE-KID
VERBO / ASSALTA
OBJECTO / BANCOS

Para lá de dump temos ainda

(descreve 'no-id)

O resultado dum chamada deste género é a descrição do nó cujo identificador é no-id, tal qual ela se obteria por (dump 'no-id), mas desta vez são escritos TODOS os nós que de alguma forma se possam atingir a partir do nó cujo identificador é no-id, a partir das suas relações descendentes.

Optou-se por NÃO REPETIR os nós que se atingem através das relações descendentes, mesmo que esses nós sejam possíveis de atingir por caminhos diversos. Esta opção só se justifica pela forma completa com que cada nó se descreve.

Caso não exista na rede um nó com identificador no-id, ocorre uma mensagem de erro alusiva a esse facto.

- Exemplo de descreve para o NODE1 (do exemplo 1) criado por build:

(descreve 'NODE1)



- 1º) mostra NODE1 exactamente como se vê acima, pela acção de dump.
2º) mostra:

Identificador: BILLY-THE-KID

Tipo: CTE

*Relações ascendentes / nó destino:
AGENTE- / NODE1*

*Relações descendentes / nó destino:
Não há relações descendentes.*

Identificador: ASSALTA

Tipo: CTE

*Relações ascendentes / nó destino:
VERBO- / NODE1*

*Relações descendentes / nó destino:
Não há relações descendentes.*

Identificador: BANCOS

Tipo: CTE

*Relações ascendentes / nó destino:
OBJECTO- / NODE1*

*Relações descendentes / nó destino:
Não há relações descendentes.*

Caso NODE1 voltá-se a apontar indirectamente, através de outro nó para, por exemplo, BANCOS, (descreve 'NODE1) não repetiria a descrição do nó BANCOS, pois uma única descrição considera-se suficientemente informativa para o utilizador poder avaliar da teia de relações que a rede no momento suporta.

4. Eliminação de conhecimento.

Para eliminar conhecimento da rede, o utilizador dispõe de



(retira 'no-id)

que vai eliminar da rede o nó cujo identificador é no-id e (eventualmente) mais alguns nós de acordo com a seguinte filosofia:

- Se o nó não existir, gera-se mensagem de erro alusiva a esse facto.

- Caso o nó exista, é eliminado de imediato.

- São eliminados os nós descendentes directos do nó eliminado, desde que esse nós não sejam asserções e só tenham relações ascendentes para o nó eliminado; assim nenhum nó asserção descendente será eliminado, e nenhum nó de nenhum tipo será eliminado se tiver relações ascendentes para outro nó distinto do nó eliminado de raiz.

- O processo de eliminação é recursivo para cada um dos descendentes directos possíveis de eliminar.

A saber: Em vez de (retira 'no-id), o utilizador poderá escrever (retira-no 'no-id). A diferença NÃO EXISTE na prática. O que se passa, é que pela primeira invoca uma macro, pela segunda invoca uma função. Eventualmente será mais eficiente a utilização da macro, pois o código correspondente não precisa de ser gerado mais do que uma vez.

Estes são no entanto pormenores do foro técnico, com os quais o utilizador não se deverá preocupar.

5. Gravação da rede em ficheiro.

Um único comando basta para gravação da rede em ficheiro:

(grava "nome do ficheiro")

O nome do ficheiro é OPCIONAL!

Isto significa que se fizer uma chamada do género (grava), gravará a sua rede (estado actual da rede) em disco num ficheiro que por defeito se chama REDE.DAT.

Se, por seu lado, resolver fazer uma chamada do género (grava "BILLY.DAT") estará a gravar a sua rede, no seu estado actual, no ficheiro BILLY.DAT.

Para mais informações sobre manutenção da rede em ficheiro, deverá consultar o aspecto 6.

6. Obtenção da rede dum ficheiro a partir dum ficheiro em disco.



Um único comando bastará para recuperar a rede para memória a partir dum ficheiro em disco:

(carrega "nome do ficheiro")

O nome do ficheiro é OPCIONAL!

Isto significa que bastará chamar (carrega) para carregamento em memória da rede. Por defeito, o ficheiro utilizado para recuperar a informação é o REDE.DAT.

Caso se faça uma chamada do tipo (carrega "BILLY.DAT"), estar-se-á a explicitar que se deseja carregar em memória uma rede tal como a descrita no ficheiro BILLY.DAT.

Para mais informação sobre manutenção da rede em ficheiro, deverá consultar o aspecto 5, na página anterior.

7. Funções várias.

As seguintes funções são as a que o utilizador pode recorrer livremente, embora não se enquadrem em nenhum dos 6 grupos de funções anteriores.

Nesta fase estão disponíveis as seguintes funções «auxiliares» :

(quantos-nos ()) - Devolve o número de nós presentes na rede.

(pertence-rede (no-id)) - Devolve t se o nó cujo identificador é recebido como argumento pertence à rede; devolve nil caso contrário.

(lista (parametro)) - O parâmetro pode ser 'nos ou 'relacoes. Se for nos, mostram-se os identificadores de todos os nós presentes na rede. Se for relacoes mostra-se a lista de todas as relações já criadas.

(ajuda) - Indica todos os comandos possíveis para o utilizador, sem no entanto fazer uma descrição dos mesmos, sendo - por isso - sempre necessária a consulta deste documento antes de utilizar a rede.

8. Shell de interacção com o utilizador.

Existe uma forma de interagir com o programa, sem ser directamente pelo Lisp. Estabelece-se uma interface de nível superior para evitar que o utilizador possa entrar instruções Lisp que não estejam directamente relacionadas com as possibilidades oferecidas pela rede.

Para activar a shell deverá escrever à prompt do lisp (RS), em que RS são as abreviaturas de Rede Semântica. A shell está presente no ficheiro Inic.Lsp e recorre a

12



uma variável global não comentada na secção das variáveis globais, pelo facto de ser utilizada só no contexto deste ficheiro; a variável em questão chama-se *comandos*.

O que faz a variável *comandos* ?

- Esta variável é uma lista que contém todas as instruções que a shell aceita. Assim, durante a interacção com a shell, são aceites os seguintes comandos:

- assercao, build, def-rel, retira, dump, descreve, lista, carrega, grava, quantos-nos?, pertence-rede?, ajuda e exit.

Todos os comandos atrás referidos já foram documentados, à excepção de ajuda, lista e exit, os quais só fazem sentido existirem (e serem comentados) no contexto da shell:

(ajuda) - Imprime todos os comandos que o utilizador pode entrar pela shell, ou seja a lista de comandos.

(lista parâmetro) - O parâmetro pode ser 'nós ou 'relações. Caso seja nós, procede-se à listagem dos identificadores de todos os nós presentes na rede. Caso contrário, isto é, recebe-se o parâmetro relações, listam-se todas as relações já definida na rede. O resultado obtido é o mesmo que ocorreria; escrevendo fora da shell, directamente no Lisp, *relacoes* -conforme descrito atrás relativamente às situações em que se interage directamente com o Lisp.

(exit) - Corta de imediato a interacção do utilizador com a shell da rede e com o próprio Lisp.

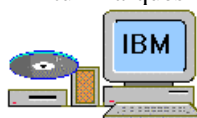
A propósito do respeitar do principio da unicidade :

Conforme se referiu desde início, a rede respeita o principio da unicidade.

Como se verifica que a rede está a respeitar o principio da unicidade ?

- Muito simplesmente, bastará criar um nó por asserção ou build e, de seguida, tentar criar exactamente o mesmo conceito. O que vai acontecer é a obtenção duma mensagem que informa o utilizador que um conceito idêntico já era suportado pela rede e que o novo conceito não foi - por isso - considerado.

No caso do conceito em questão ter sido criado por build e de seguida fizer-se a criação do mesmo conceito com sendo uma asserção, o tipo do nó é modificado para asserção.



Documentação Técnica

Esta documentação técnica é para ser considerada em conjunto com a própria listagem do programa. De facto a consulta da listagem seria por si quase auto suficiente para compreensão da actuação do programa, na medida em que cada função recorre a um nome adequado e está adequadamente comentada quanto aos parâmetros que recebe e resultados que produz, para lá de referências a pormenores do seu código.

1. Estruturas de dados utilizadas.

São declaradas duas estruturas através de defstruct e que correspondem àquilo que deverão ser um nó e uma relação. Nas estruturas a seguir descritas alguns campos são, ou têm ponteiros para outras estruturas de forma a reflectir a teia de conhecimento capturado na rede.

Eis a estrutura dum nó:

Nome-Nó
Tipo-Nó
Relações-asc
Relações-desc

O campo nome-nó destina-se a conter o identificador do nó, identificador esse que o Lisp tratará como sendo um átomo.

O campo tipo-nó pode ter um de 4 valores: Cte, Hip, True ou Var. Cada um desses valores é, internamente, um átomo que refere, respectivamente, que o nó é constante, hipótese, asserção ou variável.

O campo relações-asc deverá ser uma lista cujos elementos são estruturas do tipo relação (a seguir descritas) representativas das relações ascendentes do nó.

O campo relações-desc deverá também ele ser uma lista de estruturas representativas de relações, neste caso descendentes em relação ao nó.

A estrutura dum nó, não pode pois passar sem a estrutura duma relação. Eis a representação interna duma relação:

Nome-rel
Aponta-nó



O campo nome-rel é o nome da relação. Convencionou-se que as relações inversas têm exactamente o mesmo nome da sua relação oposta acrescido dum sinal " - " no final. Por exemplo Gosta e Gosta- podem ser os nomes de relações inversas entre si. Internamente, o nome da relação é um átomo.

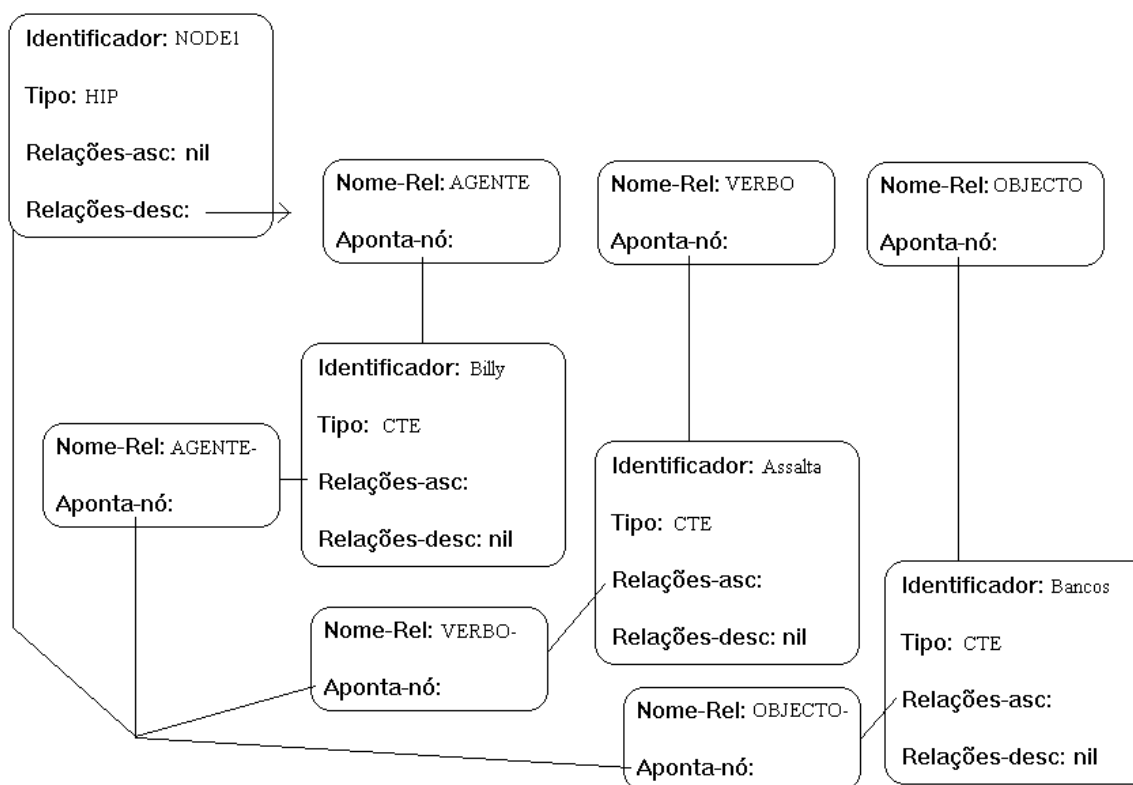
O segundo campo da relação é um nó! É um nó com uma estrutura idêntica à descrita atrás para os nós.

Muito provavelmente durante a manutenção duma rede, o mesmo nó será referido várias vezes; isto é, apontado umas vezes por relações ascendentes, outras vezes por relações descendentes com origem em nós diversos. Então - nessas ocasiões - para não se desperdiçar memória a repetir a representação do nó; e até por uma questão de naturalidade em relação à forma como guardamos, na realidade, conhecimentos, o segundo campo que estamos a discutir é um ponteiro para o nó eventualmente já criado.

Isto significa que é uma referência para a mesma zona de memória.

Na próxima página está um exemplo da representação interna que a rede faz do NODE1 descrito no exemplo 1.

Representação interna - em função das estruturas permitidas - do NODE1 mencionado no exemplo 1:



2. Variáveis globais utilizadas.

Recorre-se às seguintes variáveis globais:

rede - lista cujos elementos são estruturas nó, cada uma representando um nó presente na rede.

É esta variável que suporta a rede propriamente dita.

Escrevendo à prompt do Lisp ***rede***, devido às situações de ciclos fechados, provocada pela utilização de ponteiros, obter-se-á um ciclo infinito (enquanto a stack do sistema o suportar) em que se escreve a representação que o próprio Lisp faz da lista ***rede***.

relacoes - lista cujos elementos são os identificadores de todas as relações criadas até ao momento.

num-nos - Variável que indica o número de nós criados por build ou asserção desde o início da interação com a rede.

3. As funções e os ficheiros que as contêm.

Existem neste momento 6 ficheiros necessários à execução da rede.
São eles:

Nós.Lsp - O mais importante dos ficheiros: declara as estruturas e define as funções para criação de nós e relações.

Descreve.Lsp - Contém as funções utilizadas para a descrição de nós. Na prática, o utilizador só pode descrever nós por dump ou descreve; no entanto, essas funções recorrem a outras de nível inferior para levar a cabo o seu serviço. Todas essas funções estão neste ficheiro.

Vários.Lsp - Contém funções auxiliares e algumas macros que podem facilitar esta e seguintes fases de implementação da rede.

Grava.Lsp - Contém todas as funções relacionadas com o processo de gravação da rede num ficheiro.

Load.Lsp - Contém todas as funções relacionadas com o processo de leitura da rede dum ficheiro.

Run.Lsp - Utilizado para carregar todos os restantes. Uma vez no Lisp, bastará escrever (load 'run) para carregar todos os ficheiros em memória, e dessa forma, começar a interagir com a rede.



As várias funções presentes nos ficheiros acima descritos já estão descritas e comentadas nas listagens respectivas.

Não assumindo esta documentação um carácter de definitiva, optou-se por não repetir a informação relativa às funções e macros presentes nos diversos ficheiros. O utilizador mais interessado - dito técnico - deverá então reportar-se à leitura das listagens e respectivos comentários.

Assume-se apesar de tudo que, o leitor que proceda à leitura da listagens, tenha lido integralmente este documento, principalmente a parte relativa às estruturas de dados utilizadas e relativa a quais as funções realmente disponíveis para o utilizador.

