Universidade Federal do Paraná
Departamento de Informática

Inteligência Artificial

Representação de Conhecimento

(Aspectos gerais, Lógica e Orientação a Objetos)

Alexandre I. Direne
E-mail: alexd@inf.ufpr.br
Web: http://www.inf.ufpr.br/~alexd

BIBLIOGRAFIA

- Artificial Intelligence: A Modern Approach. Stuart Russell and Peter Novig. Terceira Ed., Prentice Hall, 2010.

- William F. Clocksin and C.S. Mellish, Programming in Prolog, Springer-Verlag, 1987

- Elaine Rich, Kevin Knight, Artificial Intelligence, Second Edition, McGraw Hill, 1993.

- Patrick H. Winston, Artificial Intelligence, Second Edition, Addison-Wesley, 1993.

SOFTWARE RECOMENDADO

Poplog System (com compilador Prolog e outros). Endereço para obtenção:

ftp://ftp.cs.bham.ac.uk/pub/dist/poplog/freepoplog.html

http://www.cs.bham.ac.uk/research/poplog/freepoplog.html

Representação de Conhecimento

Sistemas Orientados por Assertivas:

- Representação por "Encadeamento Progressivo" (REGRAS DE PRODUUÇÃO)
- Representação por "Encadeamento Regressivo" (PROLOG)
- Representação por outras Lógicas (PROPOSICIONAL, DE PREDICADOS, outras)

Sistemas Orientados a Objetos:

- Frames
- Scripts
- Redes Semânticas
- Redes de Transição (Aumentadas)

Representação de Conhecimento Orientada por incerteza

Representação por Conhecimento Procedimental

Representação por "Encadeamento Progressivo"
Sistemas de Produção

* Base (ou memória) de FATOS
* Base (ou memória) de REGRAS de PRODUÇÃO
* INTERPRETADOR DE Regras de Produção

Fatos: São assertivas presentes na memória em um dado CICLO do Interpretador de Regras.

Exemplo:

[usando camiseta] &
[usando casaco] &
…
[falta luz] &
…
[febre alta] &
[garganta inflamada] &
…
…

Regras de Produção: Uma REGRA DE PRODUÇÃO é uma declaração da forma:

SE [<CONDIÇÃO>] ENTÃO
<AÇÃO 1>;
<AÇÃO 2>;
…
…
<AÇÃO n>;
FIM-SE;

Ou ainda, uma BASE DE REGRAS é um conjunto de declarações da forma:

```
REGRA  <NOME-1>  [ <CONDICAO-1> ];
        <AÇÃO 1>;
        <AÇÃO 2>;
        …
        …
        <AÇÃO n1>;
FIM-REGRA;

REGRA  <NOME-2>  [ <CONDICAO-2> ];
        <AÇÃO 1>;
        <AÇÃO 2>;
        …
        …
        <AÇÃO n2>;
FIM-REGRA;

   .
   .
   .

REGRA  <NOME-1>  [ <CONDICAO-1> ];
        <AÇÃO 1>;
        <AÇÃO 2>;
        …
        …
        <AÇÃO nK>;
FIM-REGRA;
```

Exemplo de uma base de <u>REGRAS DE PRODUÇÃO</u> simples para calcular o Fatorial:

```
rule 1 [part_result fact ?n 1] ;
        remove([part_result fact ^n 1]);
        add([result ^n]);
endrule;

rule 2 [part_result fact ?n ?m] ;
        remove([part_result fact ^n ^m]);
        add([part_result fact ^(n*m) ^(m-1)]);
endrule;

[[part_result fact 1 7]] → database;
```

Using rule 2 with
Database: [[part_result fact 1 7]]
Conditions: [[part_result fact ? n ? m]] matching [[part_result fact 1 7]]
Actions: [remove ([part_result fact ^ n ^ m]) ;
        add ([part_result fact ^ (n * m) ^ (m -1)]) ;]

Using rule 2 with
Database: [[part_result fact 7 6]]
Conditions: [[part_result fact ? n ? m]] matching [[part_result fact 7 6]]
Actions: [remove ([part_result fact ^ n ^ m]) ;
        add ([part_result fact ^ (n * m) ^ (m -1)]) ;]

Using rule 2 with
Database: [[part_result fact 42 5]]
Conditions: [[part_result fact ? n ? m]] matching [[part_result fact 42 5]]
Actions: [remove ([part_result fact ^ n ^ m]) ;
        add ([part_result fact ^ (n * m) ^ (m -1)]) ;]

```
Using rule 2 with
Database: [[part_result fact 210 4]]
Conditions: [[part_result fact ? n ? m]] matching  [[part_result fact 210 4]]
Actions: [remove ( [ part_result fact ^ n ^ m ] ) ;
        add ( [ part_result fact ^ ( n * m ) ^ ( m -1 ) ] ) ;]

Using rule 2 with
Database: [[part_result fact 840 3]]
Conditions: [[part_result fact ? n ? m]] matching  [[part_result fact 840 3]]
Actions: [remove ( [ part_result fact ^ n ^ m ] ) ;
        add ( [ part_result fact ^ ( n * m ) ^ ( m -1 ) ] ) ;]

Using rule 2 with
Database: [[part_result fact 2520 2]]
Conditions: [[part_result fact ? n ? m]] matching  [[part_result fact 2520 2]]
Actions: [remove ( [ part_result fact ^ n ^ m ] ) ;
        add ( [ part_result fact ^ ( n * m ) ^ ( m -1 ) ] ) ;]

Using rule 1 with
Database: [[part_result fact 5040 1]]
Conditions: [[part_result fact ? n 1]] matching  [[part_result fact 5040 1]]
Actions: [remove ( [ part_result fact ^ n 1 ] ) ;
        add ( [ result ^ n ] ) ;]
```

Modelo básico para implementar um INTERPRETADOR DE REGRAS DE PRODUÇÃO:

Procedimento INTERPRETADOR(<BASE DE FATOS>) → <BASE DE FATOS>;

Repetir *INDEFINIDAMENTE* ⇐ **** CICLO DO INTERPRETADOR ****

- Avaliar as CONDICIONAIS de "cada" Regra usando os itens da Base de Fatos;
- Calcular LIMIARES de probabilidade para a propagação do efeito das Regras;
- REMOVER e/ou ADICIONAR novos fatos (corpo da Regra com condicional verdadeira);
- PARAR ciclagem no caso de nenhuma CONDICIONAL de Regra ser verdadeira.

Fim-Repetir

Fim-Procedimento;

Interpretação de Regras

versus

Execução de Seleção Múltipla em Linguagens Procedimentais

1. Em linguagens procedimentais, a ordem com que as condicionais da seleção são verificadas é FIXA. Ou seja, a primeira condicional é avaliada em seu "valor verdade"; caso seja falsa a segunda condicional é avaliada, e assim sucessivamente.
2. Em Interpretação de Regras, todas as condicionais da Base de Regras são avaliadas a cada Ciclo do Interpretador, "SEM" ordem predeterminada.
3. A cada Ciclo do Interpretador, toda a Base de Regras é "reavaliada" em vista de prováveis alterações da Base de Fatos.
4. O fim da INTERPRETAÇÃO é guiado pela "PROPAGAÇÃO" de efeitos de Regras em outras Regras por intermédio da Base de Fatos.

5. Em linguagens procedimentais, o final da EXECUÇÃO de uma seleção múltipla é regido pela ativação de "NO MÁXIMO" 1 (um) conjunto de "ações" correspondentes à "primeira condicional" VERDADEIRA.

Exemplo de uma sub-parte de uma BASE DE REGRAS:

```
…
…

rule nao_funciona [motor não funciona] ;
        vars x ;
        pr('O motor reage ?');
        readline() → x;
        add([motor ^x reage ]);
endrule;

rule falta_eletricidade [motor não reage];
        vars x ;
        pr('As luzes do automovel acendem ?');
        readline() → x;
        add([luzes acendem ^x]);
endrule;

rule falha_eletrica [luzes acendem não] ;
        [bateria descarregada] ⇒
endrule;

…
…
```

Incrementando com HEURÍSTICAS DE CONTROLE ao Interpretador de Regras:

1. Quando a condicional de mais de uma regras for verdadeira em um MESMO CICLO do Interpretador, então ATIVAR a Regra que não tiver sido ATIVADA ANTERIORMENTE (ou NO CICLO ANTERIOR, ou ainda, "RECENTEMENTE", etc...).
2. Backtracking: Se este mecanismo é ativado, o Interpretador restaura a configuração da Base de Fatos até o estado anterior a ativação da ultima regra escolhida e parte para tentar ativar uma outra (se houver).
3. Meta-Ordenação: Se este mecanismo é ativado, o Interpretador faz uso de uma ou mais META-REGRAS as quais determinam prioridades de ativação de Regras "comuns" cuja condicional é verdadeira no ciclo corrente.

Representação Orientada a Objetos

Frames

1. Representações em "Frames" (representações "explicitas" de objetos)
2. Sistema de Gerência de Frames (ver livro texto e literatura adicional)

Um frame e' uma coleção de fatos relacionados com uma entidade especifica. São extensões de representações tradicionais de fatos do tipo

<OBJETO,ATRIBUTO,VALOR>.

Assim, fatos como:

[JOAO AMA MARIA]
[JOAO AMA RENATA]
[IDADE DE JOAO 33]
[ATIVIDADE JOAO INFORMÁTICO]

podem ser reduzidos a uma representação de Frames da seguinte forma:

```
[JOAO
        [AMA
                [MARIA]
                [RENATA]
        [IDADE
                [33]
        [ATIVIDADE
                 [INFORMÁTICO]
```

Permitem maior conveniência na segmentação de uma base de dados em unidades que embutem um certo "significado," alem de oferecerem uma forma "eficiente" de armazenagem

Tais representações gozam de características de raciocínio tais como:

1. Descrição de conceitos e suas características;
2. Referencias cruzadas entre características de um conceito ou mesmo entre conceitos;
3. Cômputo dinâmico de valores de características;
4. Valores "default" de características;
5. Reaproveitamento de características e comportamentos de super-conceitos pela "herança" destas características por parte de sub-conceitos;
6. Instanciação;
7. Criterialidade;

Forma Geral

Tais representações de Frames tem a seguinte forma sintática:

```
[NOME-Frame-1
        [Caracteristica-1
                [Valor-real]
                [Valor-default]
                [Valor-calculado]
                [Valor-adicionado]
                [Valor-removido] ]

        [Caracteristica-2
                [Valor-real]
                [Valor-default]
                [Valor-calculado]
                [Valor-adicionado]
                [Valor-removido] ]
        .
        .
        .

        [Caracteristica-n
                [Valor-real]
                [Valor-default]
                [Valor-calculado]
```

[Valor-adicionado]
[Valor-removido]]]]

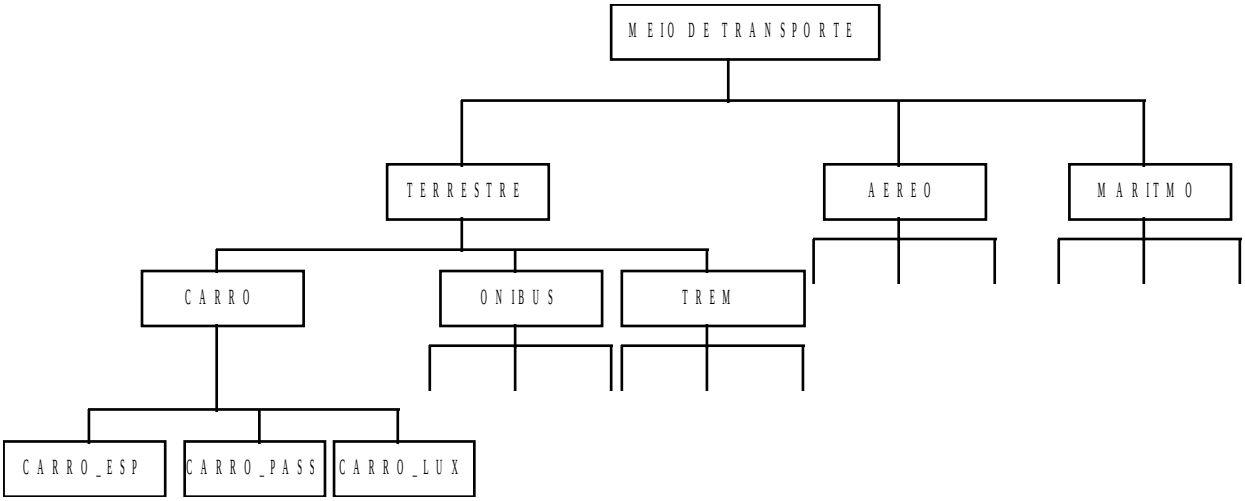
[NOME-Frame-2
… …]

…
…

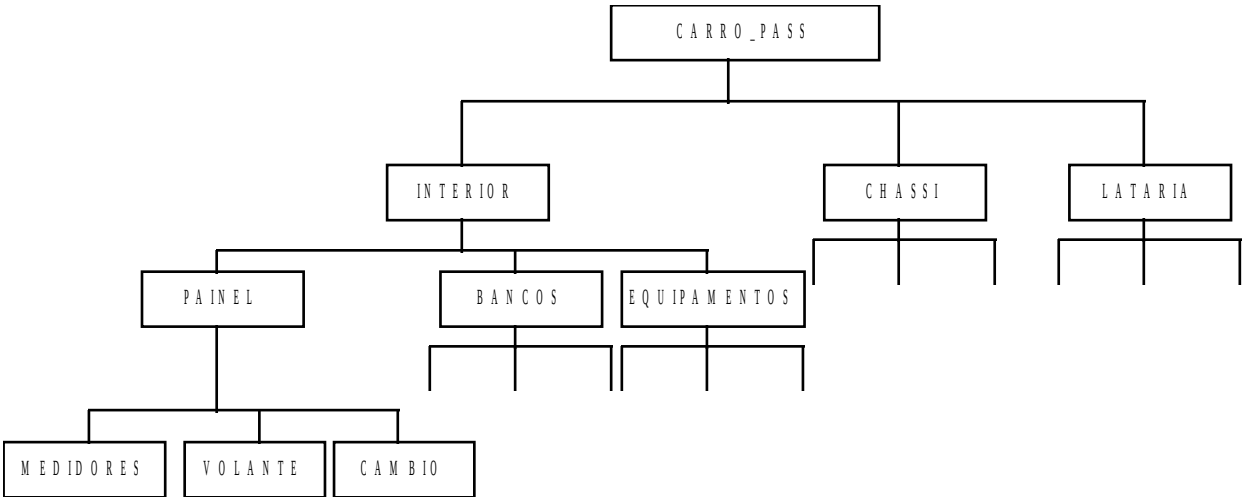
[NOME-Frame-M
… …]]

Hierarquias: Taxonomias e Partonomias

*Um exemplo de **TAXONOMIA:***



*Um exemplo de **PARTONOMIA:***



Procedimentos Básicos em um Gerenciador de Frames

Criação ou "atualização" de um frame. Caso o frame <NOME-FRAME> já exista, fput adiciona "mais" um valor (<VALOR>) para a <FACETA> de <ATRIBUTO>.

fput(<NOME-FRAME>, <ATRIBUTO>, <FACETA>, <VALOR>)

Criação ou "atualização" de um frame. Caso o frame <NOME-FRAME> já exista, fput adiciona "mais" um valor (<VALOR>) para a <FACETA> de <ATRIBUTO>.

fget(<NOME-FRAME>, <ATRIBUTO>, <FACETA>)

Retorna uma lista de valores encontrados na <FACETA> do <ATRIBUTO> do frame <NOME-FRAME>. Retorna "vazio" caso não existam valores.

fget_v_d(<NOME-FRAME>, <ATRIBUTO>)

Retorna valores se estes existirem, caso contrário retornam valores "default". Retorna "vazio" caso não existam valores default. OBS.: Não busca em super-classes de uma taxonomia.

fget_v_d_ifn(<NOME-FRAME>, <ATRIBUTO>)

Retorna valores se estes existirem, caso contrário retornam valores "default". Usa valores calculados caso existam. Retorna "vazio" caso contrário.

OBS.: Não busca em super-classes de uma taxonomia.

fremove(<NOME-FRAME>, <ATRIBUTO>, <FACETA>, <VALOR>)

Remove o <VALOR> especificado na <FACETA> de <ATRIBUTO> do frame <NOME-FRAME>.

fclamp(<NOME-FRAME-1>, <NOME-FRAME-2>, <ATRIBUTO>)

Amarra os atributos <ATRIBUTO> de dois frames distintos de forma que se um dos frames tiver seu valor alterado o outro também o terá.

fget_z(<NOME-FRAME>, <ATRIBUTO>)

Retorna valores para <ATRIBUTO> da seguinte forma: 1 - procura no domínio do frame em ordem de valor " real", "default", "calculado". Se não há sucesso, busca em super-classes. Retorna valor "real", "default", "calculado". Para no primeiro atributo não vazio.

fget_n(<NOME-FRAME>, <ATRIBUTO>)

Semelhante a fget_z exceto pelo fato de buscar em super-classes baseando-se apenas em valores reais do atributo. Caso a busca retorne vazio, fget_n volta ao frame de partida e recomeça a busca baseando-se em valores default. Finalmente, se ainda não houver valores, a busca recomeça do mesmo frame tentando valores calculados.

fget_i(. . .)

Somente busca valores reais na taxonomia/hierarquia.

fgetclasses(. . .)

Retorna uma lista de todos os super-frames acima na taxonomia.

demonfput(<NOME-FRAME>, <ATRIBUTO>, <FACETA>, <VALOR>)

Adiciona valores e ativa procs de valores adicionados nos super-frames.

demonfremove(<NOME-FRAME>, <ATRIBUTO>, <FACETA>, <VALOR>))

Remove valores e ativa procs de valores removidos nos super-frames.

Exemplos do Uso de Frames

Uma PARTONOMIA com herança !!!

```
[sp_164 [part_name [value [thorax]]]
        [i_is_a [value [sp_157]
                       [sp_160]
                       [sp_162]
                       [sp_172]
                       [sp_173]]]
        [part_type [value [root]]]
        [inst [value [marked_enl_heart]]]
        [width [value [33.2] [VISIBLE]]]]
```

Algumas coisas sobre a vida de "joao"

```
fput("joao","ama","value","maria");
fput("joao","ama","value","renata");
fput("joao","idade","value",33);
fput("joao","profissao","value","informatico");

<frame [joao [ama [value [maria] [renata]]]
        [idade [value [33]]]
        [profissao [value [informatico]]]]>

fget("joao","ama","value");
** [maria renata]

fget_v_d_f("joao","ama");
** [maria renata]

fput("joao","destreza","default","destro");
fget_v_d_f("joao","destreza")
** [destro]

fput("joao","destreza","value","canhoto");
fget_v_d_f("joao","destreza")
** [canhoto]

fpurge("joao","destreza");

<frame [joao  [ama [value [maria] [renata]]]
        [idade [value [33]]]
        [profissao [value [informatico]]]]>

fput("joao","destreza","value","destro");
fget_v_d_f("joao","destreza")
** [destro]

fput("joao","destreza","value","canhoto");
fget_v_d_f("joao","destreza")
** [destro canhoto]

fput("joao","salario","value",7000);
fput("joao","cod_imposto","value",372);
fput("joao","imposto","if_needed",[calc_imposto []]);
fget_v_d_f("joao","imposto")
** [951.2]

fclear("joao","salario","value");
fput("joao","salario","value",9500);
```

```
        fget_v_d_f("joao","imposto");
        ** [1676.2]

        fkill("joao");

        fput("joao","salario","if_needed",[ask []]);
        fput("joao","salario","range","number");
        fput("joao","salario","question",'Qual e o salário de joao ?');

        fput("joao","cod_imposto","if_needed",[ask []]);
        fput("joao","cod_imposto","range","number");
        fput("joao","cod_imposto","question",'Qual e o código de imposto de joao?');

        fput("joao","imposto","if_needed",[calc_imposto []]);

        fget_v_d_f("joao","imposto")

Qual e o salário de joao ?
? 6500
Qual e o código de imposto de joao ?
? 372
** [806.2]

        fclear("joao","salario","question");
        fclear("joao","cod_imposto","question");
        fclear("joao","salario","value");
        fclear("joao","cod_imposto","value");

        fget_v_d_f("joao","imposto")

What is the value for the salário of the joao
? 6500
What is the value for the cod_imposto of the joao
? 372
** [806.2]
```

Exemplo aplicação de raciocínio com herança de atributos de frames.

```
        fput("humano","no_pernas","value",2);
        i_fput("joao","inst","value","humano");

        fget_z("joao","no_pernas");
        ** [2]

        >>>> trace fget;
        fget_z("joao","no_pernas");

>fget joao no_pernas if_needed
<fget []
>fget joao inst value
<fget [humano]
>fget humano no_pernas if_needed          ;;; Checagem inicial de conectividade.
<fget []                                  ;;; O sistema de frames determina
>fget humano inst value                   ;;; se espera ou não respostas em
<fget []                                  ;;; forma de frames.
>fget humano is_a value
<fget []
>fget joao no_pernas value                ;;; joao tem algum valor para no.de pernas?
<fget []                                  ;;; não
>fget joao no_pernas default              ;;; tem valor "default"
<fget []                                  ;;; não
>fget joao no_pernas if_needed            ;;; seu no. de pernas pode ser calculado?
```

```
<fget []                                    ;;; não
>fget joao inst value                       ;;; joao é uma instancia de alguém?
<fget [humano]                              ;;; sim, de um humano
>fget humano no_pernas value                ;;; humanos geralmente tem numero fixo de
                                            ;;; pernas (no_pernas) ?
<fget [2]                                   ;;; yes,  2
** [2]
```

Listas de Frame:

** [robin ostrich crocodile bird reptile animal human elephant giraffe mammal lion black_mamba carnivore herbivore omnivore jumbo mary]

Frames:

```
** [bird [is_a [value [animal] [omnivore]]]
        [i_is_a [value [ostrich] [robin]]]
        [blood [value [warm]]]
        [travel [value [fly]]]]

** [lion [is_a [value [mammal] [carnivore]]]
        [gestation [value [105-110 days]]]]

** [mary [inst [value [human]]]]

** [human [is_a [value [mammal] [omnivore]]]
        [no_legs [value [2]]]
        [gestation [value [9 months]]]
        [i_inst [value [mary]]]]

** [jumbo [inst [value [elephant]]]]

** [robin [is_a [value [bird]]]]

** [animal [i_is_a [value [mammal] [reptile] [bird]]]]

** [mammal [i_is_a [value [lion] [giraffe] [elephant] [human]]]
        [is_a [value [animal]]]
        [no_legs [value [4]]]
        [blood [value [warm]]]
        [travel [value [walk]]]]

** [giraffe [is_a [value [mammal] [herbivore]]]
        [gestation [value [420-468 days]]]]

** [ostrich [is_a [value [bird]]] [travel [value [walk]]]]

** [reptile [is_a [value [animal]]]
        [i_is_a [value [crocodile] [black_mamba]]]
        [blood [value [cold]]]
        [travel [value [walk]]]]

** [elephant [is_a [value [mammal] [herbivore]]]
        [gestation [value [22 months]]]
        [i_inst [value [jumbo]]]]

** [omnivore [i_is_a [value [human] [bird]]]]

** [carnivore [i_is_a [value [lion] [crocodile] [black_mamba]]]]

** [crocodile [is_a [value [reptile] [carnivore]]]]
```

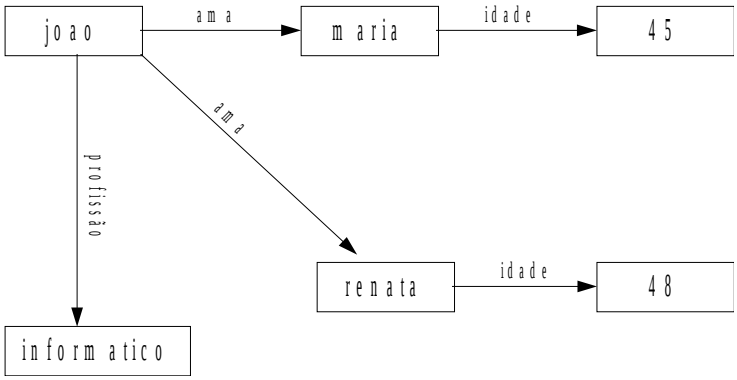
** [herbivore [i_is_a [value [giraffe] [elephant]]]]

** [black_mamba
[is_a [value [reptile] [carnivore]]]
[travel [value [slither]]]]

Redes Semânticas

Também conhecidas como *Redes de Associação.* Descrevem associações entre conceitos ou "entidades" de um domínio. Tais redes são representadas como Grafos Dirigidos ("nodos" e "ligações") onde os nodos constituem os conceitos do domínio e as ligações representam as relações entre os conceitos.

Uma ligação deve ser rotulada com o mnemônico da relação. Por exemplo, se construirmos uma rede semântica a partir de "joao" teremos:



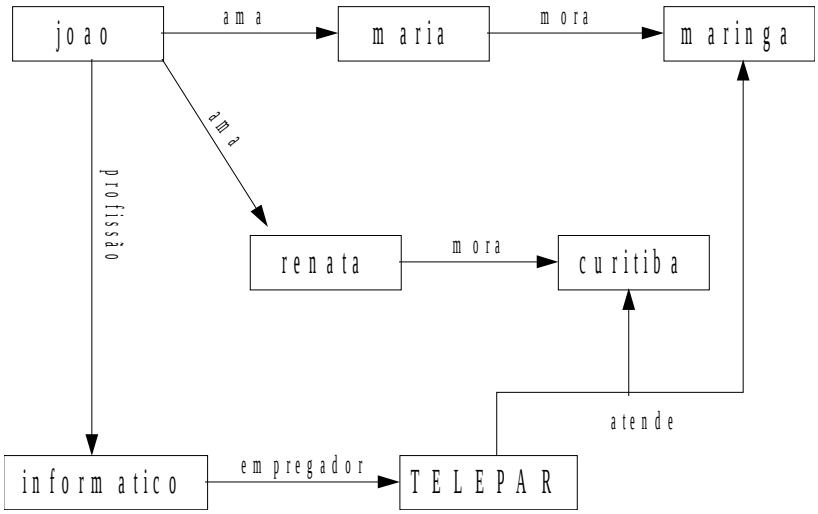
Redes Semânticas se tornaram métodos populares entre pesquisadores de Inteligência Artificial pois permitem a construção de representações de conhecimento a partir de formas gráficas. Tais redes são apenas diagramas da relação factual que escolhemos para "intencionalmente" representar em uma base de conhecimento. Assim, este formalismo pode ser facilmente encarado como forma de se descrever um estado de "crenças" que retrata, em um dado momento, as condições do "mundo" (domínio).

Do ponto de vista de implementação, redes semânticas podem utilizar "Frames" como método "interno" de representação computacional para o aspecto gráfico "externo".

Buscas Complexas

Quando se aborda um domínio de informação com um grafo de conceitos interligados, a idéia de "caminhos" de informação se sobressai. Iniciando em um nodo particular do grafo, pode ser imaginado um "atalho" ao longo de uma serie de relacionamentos para que se determine um conjunto de conceitos relacionados, de acordo com uma dada "distancia" do nodo (ou conceito) de origem. A partir daí, é possível se fazer consultas "complexas" ou até mesmo tirar "conclusões" das consultas feitas.

Como ilustração, considere o seguinte exemplo de Rede Semântica:



Pode-se observar o seguinte a respeito de joao:

* suas relações afetivas;
* as idades das pessoas que joao ama;
* os locais onde moram estas pessoas;
* os locais de serviços onde se empregador atende;

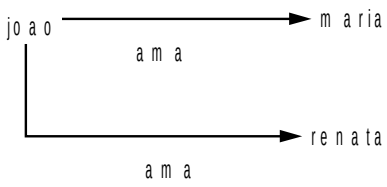
Exemplos de Busca

A "simplicidade visual" das Redes Semânticas só pode ser avaliada por seres humanos pois são equipados com capacidade de reconhecimento de imagem. A "visualização" de conceitos e relações do ponto de vista computacional deve ser efetuada por intermédio de avaliação "textual" da informação.

Como exemplos de consultas complexas, apoiado por um procedimento de busca hipotético denominado c_fget_z, temos:

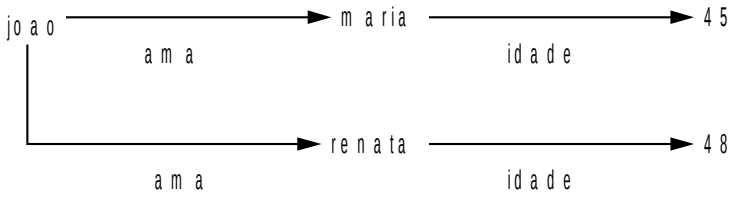
Quais as pessoas que joao ama ?

c_fget_z([[joao] ama]);



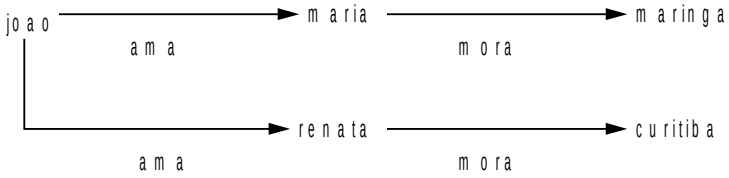
Qual a idade de todas as pessoas que joao ama ?

c_fget_z([[[joao] ama] idade]);



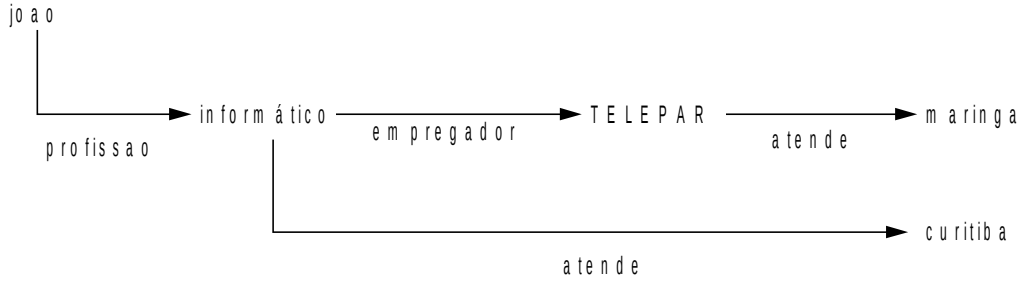
Quais os locais em que joao mantém relações afetivas ?

c_fget_z([[[joao] ama] mora]);



Quais os locais onde o empregador de joao atende com seus serviços ?

c_fget_z([[[[joao] profissao] empregador] atende]);



Quais os empregadores de joao e alberto ?

c_fget_z([[[joao alberto] profissao] empregador]);