Universidade do Minho
Escola de Engenharia
Departamento de Informática
Licenciatura em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

Trabalho de Grupo – 1º Exercício

Programação em Lógica

Braga, Abril de 2014


David Manuel de Sá de Angelis
Nº60990


Sérgio Lucas dos Santos Oliveira
Nº61024


Marcos André Oliveira Ramos
Nº61023

Resumo

Este trabalho consiste no desenvolvimento de um sistema de representação de conhecimento e raciocínio que permita caracterizar um universo de discurso com o qual se pretende descrever um cenário.

O cenário pode ser visto como um conjunto de pontos coordenados num gráfico ortogonal onde cada ponto representa um certo local ou serviço. Assim, recorrendo a certos conhecimentos de trigonometria e teoria de grafos deverá ser possível realizar operações como calcular distâncias entre pontos, identificação de pontos e correspondentes serviços, distancia mínima para percorrer uma sequência de pontos, etc...

A realização do exercício permitiu ao grupo conhecer e aplicar algumas funcionalidades da linguagem de programação em lógica PROLOG, permitindo um melhoramento na capacidade de utilização da ferramenta SICStus Prolog.

Índice

Introdução

Com vista a desenvolver os nossos conhecimentos de representação de conhecimentos e raciocínio foi-nos proposto pelos professores de SRCR(Sistemas de Representação de Conhecimento e Raciocínio) o desenvolvimento, em linguagem PROLOG, de um sistema capaz de descrever um cenário geográfico de gestão de informação. Este cenário consiste num conjunto de pontos coordenados num espaço bidimensional, que representam a localização de serviços numa região.

Posto isto, o grupo decidiu que seria de interesse representar informação sobre a produção e o consumo de energia de uma "cidade", sendo que numa cidade existem, naturalmente, produtores e consumidores de energia. Isto é, cada nodo representado no nosso sistema irá conter informação não só relativa à sua posição, mas também ao seu consumo/produção de energia.

Tendo a nossa "cidade" representada, é necessário que o sistema a desenvolver possua um conjunto de funcionalidades que permitam saber coisas sobre a informação representada. Essas funcionalidades correspondem ao que é pedido no enunciado deste exercício e foram implementadas na integra pelo grupo.

Para a realização deste trabalho foi necessário recorrer ao calculo trigonométrico e teoria de grafos. A interacção com a base de conhecimentos será feita em linguagem JAVA recorrendo à biblioteca JASPER.

Descrição do Trabalho e Análise de Resultados

Interpretação do problema

Após a leitura e interpretação do enunciado, o grupo decidiu que seria interessante representar, ainda que de forma muito simplista, a rede energética de uma cidade. Segundo a nossa interpretação, existem 3 tipos de actores numa rede energética: produtores, consumidores e produtores-consumidores. Assim, no contexto do nosso problema, decidimos representar esses três papeis pelas seguintes entidades:

- Central eléctrica (produtor)
- Casa (consumidor)
- Prédio (produtor-consumidor)

Em termos da sua representação, estas entidades são bastante semelhantes, sendo que o único aspecto que as distingue é o factor de produção/consumo. No nosso sistema, uma central apenas produz, uma casa apenas consume e finalmente, o prédio apresenta obrigatoriamente consumo e produção.

Estas características são muito rígidas e restritas sendo que qualquer alteração nos seus valores poderá fazer com que o nodo passe de um tipo para outro. Isto é, na eventualidade de por exemplo, um prédio, deixar de produzir electricidade, passava para todos os feitos a ser considerado uma casa.

Para além dos valores de consumo e/ou produção, cada nodo contem ainda um nome e a sua localização no eixo cartesiano, naturalmente.

Em termos de contextualização, a distancia(tanto trigonométrica como o caminho mais curto) entre os nodos pode ser vista como o custo da electricidade fluir por entre os diversos nodos.

Base de Conhecimento

Para prosseguir com a realização do nosso trabalho prático, foi então necessário definir um conjunto de predicados que nos permitisse representar o problema contextualizado em cima.

O predicado que serve como base de todo o nosso trabalho é o predicado **nodo**.

Representação dos nodos

Extensão do predicado Nodo:Nome,X,Y,Producao,Consumo -> {V,F}

```
%Casas
nodo(a,3,2,0,2).
nodo(b,1,5,0,3).
nodo(c,10,8,0,5).
%Centrais
nodo(d,2,10,7,0).
nodo(e,7,4,5,0).
%Predios
nodo(f,5,6,1,4).
nodo(g,6,1,2,3).
```

Este predicado nodo representa um ponto na "cidade" e contem informação sobre o nome do nodo, as suas coordenadas X,Y assim como a sua quantidade de energia produzida e consumida. No nosso trabalho, temos 7 pontos representados: 3 casas, 2 centrais e 2 prédios. Estão apenas representados 7 pontos para facilitar a compreensão do problema e da nossa resolução, no entanto se quiséssemos representar mais nodos seria perfeitamente possível de o fazer, pois não alterava o raciocínio, apenas a quantidade de informação a representar. Na nossa interpretação, apenas é considerado um quadrante e todos os nodos se encontram aí representados (1º quadrante).

Tendo em conta os nodos acima representados, ficamos com a seguinte rede:

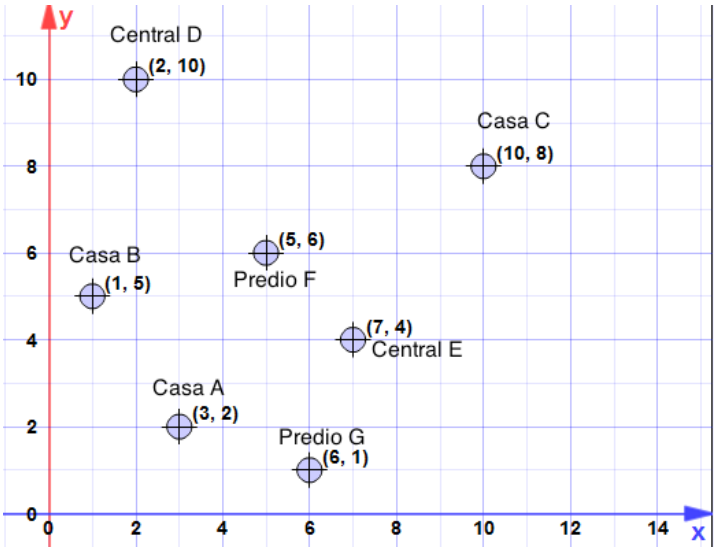

Figura 1 - Topologia da cidade (sem arcos)

Tendo os nodos representados, foi necessário criar predicados para que possamos interrogar a nossa base de conhecimento. Começamos então por criar 4 predicados para conseguirmos saber qual o tipo de um nodo.

Extensão do predicado Casa:Nodo-> {V,F}

```
casa(nodo(N,X,Y,P,C)):-P==0,C>0.
```

Um nodo é considerado uma casa unicamente se a sua produção de energia for zero e o seu consumo for maior do que zero, ou seja, uma casa é exclusivamente consumidora.

Extensão do predicado Predio:Nodo-> {V,F}

```
predio(nodo(N,X,Y,P,C)):-P>0,C>0.
```

Um nodo é considerado um prédio unicamente se tanto a sua produção como consumo de energia forem maiores do que zero.

Extensão do predicado Central:Nodo-> {V,F}

```
central(nodo(N,X,Y,P,C)):-P>0,C==0.
```

Um nodo é considerado uma central unicamente se a sua produção de energia for superior a zero e o seu consumo for estritamente zero, ou seja, uma central é exclusivamente produtora.

Por fim, temos um predicado que faz uso dos três predicados acima descritos, dizendo qual o tipo de um determinado nodo.

Extensao do predicado tipoNodo:Nodo,tipo -> {V,F}

```
tipoNodo(N,casa):-casa(N).
tipoNodo(N,predio):-predio(N).
tipoNodo(N,central):-central(N).
```

Depois de termos os nossos nodos representados, foi necessário implementar a conectividade entre esses mesmos nodos. Para tal, implementamos dois predicados: arco e connected.

Ligações entre nodos

Extensão do predicado Arco:Nodo,Nodo -> {V,F}

```
arco(nodo(a,3,2,0,2),nodo(b,1,5,0,3)).
arco(nodo(b,1,5,0,3),nodo(d,2,10,7,0)).
arco(nodo(d,2,10,7,0),nodo(f,5,6,1,4)).
arco(nodo(f,5,6,1,4),nodo(c,10,8,0,5)).
arco(nodo(c,10,8,0,5),nodo(e,7,4,5,0)).
arco(nodo(e,7,4,5,0),nodo(g,6,1,2,3)).
arco(nodo(g,6,1,2,3),nodo(a,3,2,0,2)).
arco(nodo(a,3,2,0,2),nodo(f,5,6,1,4)).
arco(nodo(f,5,6,1,4),nodo(e,7,4,5,0)).
```

Extensão do predicado Connected:Nodo,Nodo,D -> {V,F}

```
connected(X,Y) :- arco(X,Y).
connected(X,Y) :- arco(Y,X).
```

Esta conectividade entre nodos funciona em duas etapas distintas. Primeiramente, dizemos que existe um arco entre dois nodos, sendo que na nossa rede existem nove arcos. Contudo, dizer arco(A,B) não é o mesmo que dizer que A e B estão ligados. Para tal, é necessário recorrer ao predicado connected que diz que A e B estão ligados se existe um arco de A para B ou de B para A, ou seja, a conexão entre arcos funciona nos dois sentidos. Segue-se então a topologia da cidade com as ligações entre os nodos representadas.

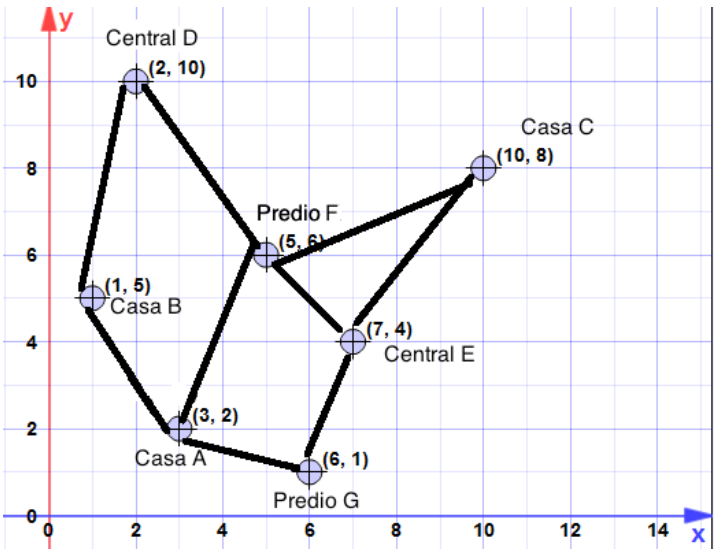

Figura 2 - Topologia da cidade (com ligações)

Seguindo a linha do enunciado, era necessário implementar o predicado que calculasse a distancia entre pontos. Temos então o seguinte predicado distancia:

Distancias entre nodos

Extensão do predicado Distancia:Nodo,Nodo,D -> {V,F}

```
distancia(nodo(N,X,Y,P,C),nodo(N2,X2,Y2,P2,C2),D):-D is
sqrt(exp((X2-X),2)+(exp((Y2-Y),2))).
```

Naturalmente, este predicado calcula a distancia trigonométrica entre os dois nodos recebidos recorrendo ao famoso "Teorema de Pitágoras", que diz que o quadrado da hipotenusa é qual à soma do quadrado dos catetos. Para implementar este teorema,

recorremos ao prolog, onde a raiz quadrada é dada por "sqrt(valor)" e a exponenciação é dada por "exp(base,expoente), como indicado pelo código na figura acima. É importante realçar que este predicado distancia calcula **apenas e só** a distancia trigonométrica entre dois nodos, não havendo aqui qualquer relação com as ligações dos nodos. Isto significa que é possível obter a distancia entre dois nodos mesmo que não exista qualquer ligação ou caminho entre elas.

Segue-se então o problema de encontrar pontos numa determinada região. Uma vez que o enunciado não era explicito que tipo de região era pretendida, o grupo decidiu implementar um conjunto de predicados que devolvessem que pontos se encontravam num rectângulo abstracto passado e definido pelo utilizador. Mais uma vez, este problema é resolvido em dois passos usando um predicado auxiliar chamado "região" e outro chamado "encontrapontosregiao".

Regiões de nodos

Extensão do predicado Regiao:Coordenadas,Coordenadas,Coordenadas,Coordenadas,Nodo-> {V,F}

```
regiao((X0,Y0),(X1,Y1),(X2,Y2),(X3,Y3),nodo(N,X,Y,P,C)):-
nodo(N,X,Y,P,C),X>=X0,Y=<Y0,X=<X1,Y=<Y1,X=<X2,Y>=Y2,X>=X3,Y>=Y3.
```

Extensao do predicado EncontraPontosRegiao: Coordenadas, Coordenadas, Coordenadas ,Coordenadas, [Nodos]-> {V,F}

```
encontrapontosregiao((X0,Y0),(X1,Y1),(X2,Y2),(X3,Y3),S):-
findall(nodo(N,X,Y,P,C),regiao((X0,Y0),(X1,Y1),(X2,Y2),(X3,Y3),n
odo(N,X,Y,P,C)),S).
```

Em ambos os predicados, as quatro coordenadas passadas como argumento definem o rectângulo da seguinte forma: a primeira coordenada indica o canto superior esquerdo do rectângulo e as restantes três definem os restantes cantos no sentido dos ponteiros do relógio.

O predicado região é bastante simples e apenas faz comparações entre as coordenadas X,Y do rectângulo e de um dado nodo para dizer se esse nodo se encontra ou não na região definida, sendo que se o nodo se se encontrar na fronteira do rectângulo, é considerado que pertence à região.

Por outro lado, o predicado "encontrapontosregiao" faz uso do "findall" disponibilizado no prolog, para encontrar todas as soluções que satisfazem um dado predicado. Neste caso, queremos todos os pontos que satisfazem o predicado "região", que corresponde obviamente a todos os pontos que se encontram dentro da dada região passada como parâmetro.

O próximo problema a ser proposto era o do calculo da distancia necessária para percorrer um determinado conjunto de pontos. Temos então o predicado "distpontos":

Calculo com litas de nodos e caminhos

Extensão do predicado DistPontos:[nodos],R -> {V,F}

```
distpontos([nodo(N,X,Y,P,C)],0).
distpontos([nodo(N,X,Y,P,C),nodo(N1,X1,Y1,P1,C1)|T],R):-
distancia(nodo(N,X,Y,P,C),nodo(N1,X1,Y1,P1,C1),D),distpontos([nod
o(N1,X1,Y1,P1,C1)|T],R1),R is R1+D.
```

Ao receber uma lista de pontos, este predicado executa o seguinte raciocínio:

- Se a lista só tiver um ponto a distancia dada será 0 (caso de paragem).
- Se a lista tiver mais do que um ponto, será calculada a distancia entre os dois primeiros pontos da lista e continuaremos recursivamente o calculo para o resto da lista, **eliminando a cabeça actual da lista.** Eventualmente o calculo de paragem será atingido.

Por ultimo, temos o desafio mais difícil deste enunciado que envolvia calcular a distância mínima necessária para percorrer um determinado conjunto de pontos, o que corresponde na prática a calcular o caminho mais curto entre um conjunto de nodos, onde a distancia representa o custo de levar a electricidade entre os nodos.

Para resolver este problema foi necessário recorrer a uma série de predicados que nos permitisse fragmentar o problema maior e mais complexo e problemas mais pequenos e simples. Assim, foram desenvolvidos os predicados "path", "travel", "caminhos", "min" e "cmc". Adicionalmente, temos os predicados "concatenar" e "inverter", mas estes são apenas auxiliares e não contribuem directamente para a resolução do problema de caminho mais curto.

Segue-se então o predicado "path". Este predicado devolve-nos um caminho possível entre dois nodos A e B passados como argumento, partindo de A (assumindo que A foi passado primeiro como argumento).

Extensão do predicado Path: nodo,nodo,[nodo] -> {V,F}

```
path(A,B,Path):-travel(A,B,[A],Q),inverter(Q,Path).
```

O que este predicado "path" faz é recorrer ao predicado travel e de segunda inverter o resultado devolvido por este, de forma a por os nodos pela ordem correcta.

Extensão do predicado travel:nodo,nodo,[nodoVisitado][nodo] -> {V,F}

```
travel(A,B,P,[B|P]) :-connected(A,B).
travel(A,B,Visited,Path) :-connected(A,C),C \==
B,\+member(C,Visited),travel(C,B,[C|Visited],Path).
```

O predicado "travel" permite então percorrer o grafo à procura dos caminhos, sendo que cada vez que passa por um nodo, esse mesmo é inserido na lista do nodos já visitados de forma a evitar ciclos no grafo.

Assim, o predicado" travel" recebe um nodo inicial, um nodo final, uma lista para os nodos visitados, e uma lista com os nodos a percorrer para chegar ao nodo final.

- Caso exista uma ligação direta entre o nodo inicial e o nodo final, o nodo final é inserido na lista dos nodos que constituem o caminho.
- Caso não existe ligação directa procuramos um nodo intermedio ligado ao nodo inicial, esse nodo intermedio tem de ser diferente do nodo final e ainda não visitado.
- Recursivamente chamamos a função travel mas desta vez o nodo intermedio é inserido na lista dos nodos visitados e utilizado como nodo inicial.

Extensão do predicado caminhos:nodo,nodo,[nodoCaminho],Custo -> {V,F}

```
caminhos(nodo(N,X,Y,P,C),nodo(N1,X1,Y1,P1,C1),Caminho,Custo):-
path(nodo(N,X,Y,P,C),nodo(N1,X1,Y1,P1,C1),Caminho),distpontos(Ca
minho,S), Custo is S.
```

De seguida temos o predicado "caminhos", que dados dois nodos, consegue obter **um qualquer caminho** entre esses mesmos nodos e fazer a associação necessária entre o caminho, e o custo de percorrer esse mesmo caminho. Isso é conseguido fazendo uso dos predicados "path" e "distpontos". Primeiramente, é utilizado o predicado "path" para obtermos um caminho possível entre os dois nodos. De seguida, e **sabendo que temos agora um caminho válido e existente,** é chamado o predicado "distpontos" para nos dar o somatório da distancia entre os vários pontos do caminho.

Extensão do predicado min: [(Caminho,Custo)],Caminho,Custo -> {V,F}

```
min([(X,Y)],X,Y).
min([(X,Y),(X1,Y1)|T],L,C):-Y =< Y1, min([(X,Y)|T],L,C).
min([(X,Y),(X1,Y1)|T],L,C):-Y > Y1, min([(X1,Y1)|T],L,C).
```

O predicado "min" tem como única função determinar qual o caminho mais curto e o seu custo, partindo de uma lista de tuplos(Caminho, Custo) . É executado o seguinte raciocínio:

- Se a lista só tiver um tuplo, será esse o menor par (caso paragem).
- Se a lista tiver mais do que um tuplo, serão comparados os dois primeiros tuplos. O tuplo cujo custo for maior será eliminado da lista.
- O processo será repetido até restar apenas um elemento na lista e temos o caso de paragem.

Extensão do predicado cmc:nodo,nodo,[Caminho],Custo -> {V,F}

```
cmc(nodo(N,X,Y,P,C),nodo(N1,X1,Y1,P1,C1),CamFinal,CustMin):-
findall((Caminho,Custo),caminhos(nodo(N,X,Y,P,C),
nodo(N1,X1,Y1,P1,C1),Caminho,Custo),S),min(S,CamFinal,CustMin).
```

F n”
e da funcionalidade prolog “findall” para, de facto, determinar qual o caminho mais curto entre dois nodos. Em primeiro lugar, são calculados todos os caminhos possíveis entre dois nodos, aplicado o “findall” ao “cmd”, especificando que queremos que as soluções sejam devolvidas na forma (Caminho, Custo). Com isto, irá ser obtida uma lista com a forma [(Caminho, Custo)], que como vimos em cima, é precisamente o que o predicado “min” precisa para determinar o caminho mais curto. Ficamos assim com o problema resolvido.

Conclusão

Este trabalho prático foi realizado com a intenção de solidificar os conhecimentos de representação de conhecimento e raciocínio.

Foram aplicados os conhecimentos de prolog que adquirimos ao longo das aulas, ao mesmo tempo que esses mesmos conhecimentos foram continuamente desenvolvidos à medida que o trabalho ia sendo realizado, uma vez que foi necessário resolver um conjunto de novas situações e problemas.

Em geral, penso que o nosso grupo conseguiu analisar e resolver prontamente a maioria dos problemas apresentados neste enunciado. No entanto, é importante deixar a nota que o grupo debateu-se no predicado "arco" antes da introdução do predicado "connected", pois estávamos a gerar um ciclo infinito ao querer representar os arcos válidos nos dois sentidos. No que toca aos predicados para resolver o caminho mais curto, houve alguma hesitação de como selecionar o caminho mais curto após termos a lista de todos os caminhos possíveis. No fim, todas essas dificuldade conseguiram ser ultrapassadas e o grupo respondeu na integra a tudo o que foi proposto neste enunciado prático.

No futuro, esperamos que os conhecimentos aqui desenvolvidos nos sejam úteis para conseguirmos implementar soluções de representação de conhecimento e raciocínio mais complexas e em contextos mais úteis.