LAB2 – Laboratório 2 de Inteligência Artificial
Assunto: Prolog

I - Observações:

1. **O trabalho é em Dupla:** É permitido discutir com outras duplas os problemas e as estratégias para solucioná-los, mas as implementações das soluções, bem como as resoluções dos exercícios propostos devem ser feitas somente pelos componentes da dupla. Trabalhos plagiados da Internet ou de outras duplas não serão corrigidos e receberão nota 0 (zero).

2. **Forma de Entrega:** Entrega Eletrônica via **SINEF**. O relatório deve ser entregue em formato PDF (outros formatos não serão corrigidos). Todos os arquivos fonte e o relatório devem ser enviados em um único arquivo ZIP (*não é RAR, nem TAR, etc!*) cujo nome deve ser: ***PrimeiroNome_UltimoNome_LAB1.zip*** onde ***PrimeiroNome*** e ***UltimoNome*** são o primeiro e o último nome de algum dos membros da dupla. **Qualquer entrega fora deste padrão será penalizada em 20% da nota.**

3. **Data Limite para Entrega: 05/05/2010 (Quarta-feira) até as 23:59 pelo SINEF**

II - O que deve ser entregue:

1. **Um relatório contendo:**
   a) os nomes dos componentes da dupla;
   b) os resultados obtidos com a execução dos programas das tarefas propostas;
   c) análise dos resultados obtidos.

2. **Todos os códigos fontes necessários para execução do programa em Prolog criado para solução das tarefas.**

3. **ATENÇÃO:** Não é preciso entregar o programa gerado a partir da execução do tutorial (item III). Apenas deve ser entregue o programa criado para resolver as tarefas propostas (item IV).

III – Tutorial Básico de Prolog

Faça o download da biblioteca tuProlog, que pode ser encontrada no endereço: **http://alice.unibo.it/xwiki/bin/view/Tuprolog/**. É necessário fazer o download somente da última versão do arquivo *jar.* No momento confecção deste texto, o arquivo jar da última versão do tuProlog era *2p-3.0-alpha.jar*. Execute as seguintes sub-tarefas para familiarização com o ambiente

O tuProlog possui uma IDE gráfica para teste de programas escritos em Prolog. Abra o *prompt* de comando do Windows (ou console no Linux) e vá para a pasta na qual você fez o *download* do arquivo *jar* do tuProlog. Execute a interface gráfica utilizando o seguinte comando:

```
java -cp 2p-3.0-alpha.jar alice.tuprologx.ide.GUILauncher
```

Depois de executado este comando, uma janela como a da figura 1 aparecerá. Na janela, a caixa de texto na parte de cima mostra o programa em Prolog que está ativo. A partir deste ponto, esta caixa de texto será referida como *caixa de programa* ou simplesmente *programa*. Uma segunda caixa de entrada de texto, marcada com o símbolo “?-” está logo abaixo da *caixa de programa*. Esta caixa

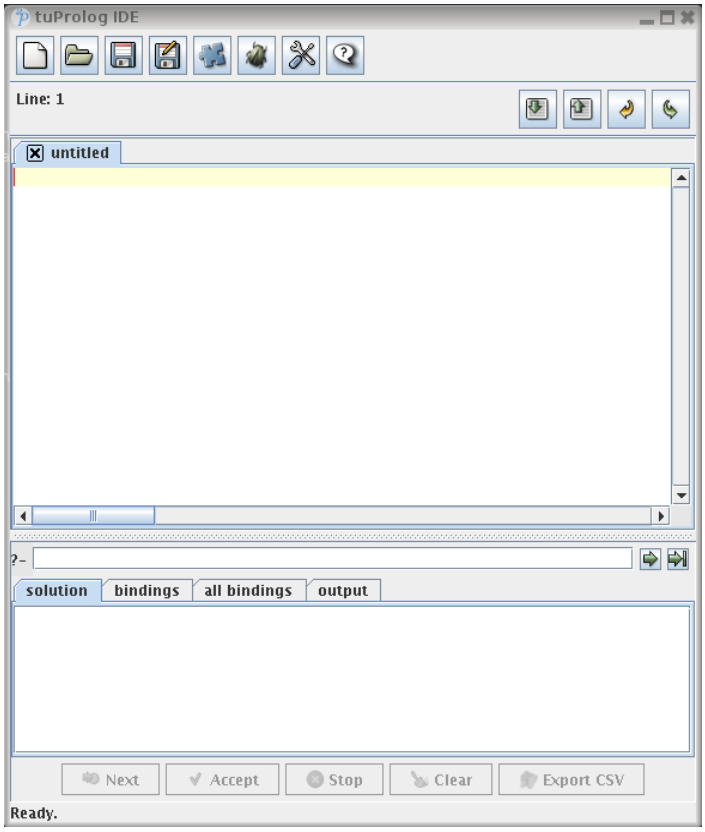


Figura 1: IDE do tuProlog

será chamada de *prompt* de agora em diante. A utilidade das abas abaixo do *prompt* será detalhada oportunamente.

No *prompt*, entre com o seguinte texto:

```
ensolarado.
```

Pressione a tecla *ENTER* ou a seta simples para esquerda, que está posicionada à esquerda do *prompt* (o *tooltip* que aparece quando o *mouse* está sobre a seta é "*solve*"). Qual o resultado desta execução na aba *solution* da caixa de mensagens abaixo do *prompt?*

FATOS SIMPLES

Fatos, em Prolog, devem ser iniciados por letra minúscula e não podem conter espaços. Fatos são verdades que podem ser consultadas por meio de sentenças escritas no *prompt*. Os fatos são entrados na *caixa de programa* da IDE.

Entre com os seguintes fatos na sua *caixa de programa*. Depois de entrar com estes fatos, clique no botão com uma seta verde apontada para cima, cujo *tooltip* indica "*Set Theory*". Fazendo isso, você está entrando com uma nova base de conhecimento para o Prolog.

```
ensolarado.
fulano_tem_gripe.
```

No *prompt* digite "ensolarado." e aperte ENTER. Qual o resultado desta execução na aba *solution?* Depois digite "chuvoso." no *prompt* e aperte ENTER. Qual o resultado desta execução na aba *solution*?

FATOS COM ARGUMENTOS

Fatos podem conter vários argumentos. Fazendo um paralelo com a lógica de primeira ordem (LPO) vista em sala de aula, fatos com argumentos podem ser vistos como relações unárias, binárias, etc, entre objetos.

Entre com os seguintes fatos com argumentos na sua *caixa de programa* e clique no botão para aceitar a nova base de conhecimento, como foi feito com fatos sem argumentos.

```
come(fulano, abacaxi).
come(fulano, couve).
come(fulano, tomate).
come(ciclano, couve).
```

No *prompt* digite “come(fulano,couve).” e aperte ENTER. Qual o resultado desta execução na aba *solution?* E qual o resultado se for realizada a consulta “come(fulano,cereja).”?

VARIÁVEIS

Consultas mais interessantes do que as apresentadas até agora podem ser feitas utilizando o conceito de variáveis. Variáveis em Prolog iniciam-se com uma letra maiúscula. Utilizando a mesma base de conhecimento entrada anteriormente, pode-se fazer a consulta sobre o que *ciclano* come, utilizando o seguinte comando no *prompt*:

```
come(ciclano, X).
```

Execute este comando no *prompt* de sua IDE e verifique o resultado. Depois, execute o comando *“come(fulano,X)”*. Você notará que os botões *Next* e *Accept* ficarão habilitados depois do comando. Clicando no botão *Next* a próxima solução para a consulta é mostrada na aba *solution*. Cliando em *Accept* você aceitará o valor mostrado na aba *solution* como novo valor para a variável X e terminará a consulta. O valor da variável pode ser visto na aba *bindings*.

Pode-se combinar o resultado da consulta sobre o que come *fulano* com a consulta sobre o que come *ciclano*. Para isto, utiliza-se uma conjunção, representada por uma vírgula em Prolog. Para listar os alimentos que *fulano* come combinados com os que *ciclano* come, basta-se executar o seguinte comando no *prompt*.

```
come(fulano,X),come(ciclano,Y).
```

Note que um algoritmo de *backtracking* é executado quando se faz esta consulta. Primeiro, o Prolog fixa o valor de X, variando o valor de Y para todos os possíveis. Quando termina os valores de Y possíveis, o Prolog irá para o proximo X e iniciará novamente a variação dos valores de Y. Execute o código para verificar.

Outra possibilidade é que esteja-se interessando apenas nos alimentos que são comuns nas dietas de *ciclano* e *fulano.* Para este caso, pode-se utilizar o operador “=”, como visto em LPO. Utilizando o comando a seguir, os alimentos que são iguais nas dietas de *fulano* e *ciclano* serão mostrados como solução.

```
come(fulano,X),come(ciclano,Y),X=Y.
```

Ou, simplesmente: `come(fulano,X),come(ciclano,X).`

REGRAS

Além dos fatos, em Prolog pode-se implementar regras que simbolizam condições. Por exemplo, pode-se inferir que Sócrates é mortal, se soubermos que Sócrates é humano e existir a regra "Alguém é humano → Alguém é mortal". No Prolog, o conectivo condicional (→) é representado pelo símbolo "**:**-". No entanto, a sintaxe do Prolog inverte a direção da condição para facilitar a leitura das definições. Assim, para implementar a regra "Alguém é humano → Alguém é mortal" em Prolog, seria criada uma regra da seguinte forma:

```
mortal(Alguem) :- humano(Alguem).
```

Para facilitar o entendimento da regra, pense no símbolo ":-" como sendo "se". Assim, a regra é lida corretamente como "Alguém é mortal se Alguém é humano". Note que "Alguem", na regra do Prolog, começa com letra maiúscula. Isto é devido ao fato de que "Alguem" é uma variável e deve começar com letra maiúscula segundo as regras sintáticas da linguagem.

Para testar esta forma simples de implicação, adicione a regra acima à sua *caixa de programa* e inclua o fato "humano(socrates).". Depois, faça a consulta "mortal(socrates)." no *prompt*. Você notará que, mesmo sem dizer explicitamente que "mortal(socrates)" é verdadeiro, o Prolog foi capaz de inferir isto a partir da regra que você criou.

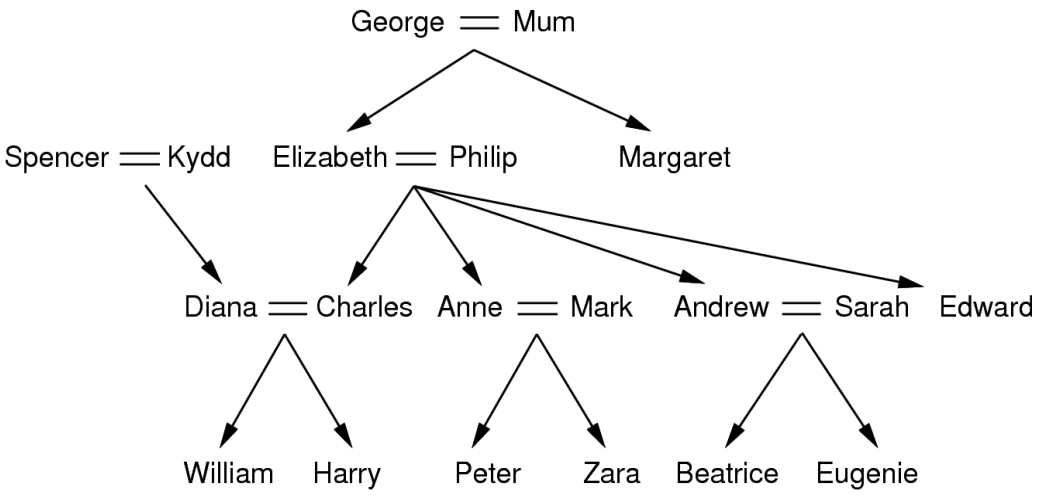
O lado esquerdo de uma regra é chamado de *cabeça* (ou conclusão), enquanto o lado direito é chamado de *corpo* (ou condição). No *corpo* podem ser usados conectivos lógicos simbolizando as operações de E (,), OU (;) ou NÃO (not( )). Assim, para dizer que o salão (X,Y) é seguro se não tem nenhum wumpus e se não tem poços, pode-se utilizar a regra:

```
seguro(X,Y) :- not(wumpus(X,Y)), not(poco(X,Y)).
```

IV – Tarefas

1. Escreva um programa em Prolog que contenha, além dos fatos a seguir, uma única regra que permita inferir o fato f*eminino*(l*aura*):
   - *masculino*(*joao*).
   - *conjuge(joao,laura).*

2. Escreva fatos para as relações *genitor(G,F), conjuge(X,Y), feminino(Y)* e *masculino(X)* de acordo com as relações de parentesco mostradas na figura abaixo. Considere que o primeiro parâmetro da regra *genitor* é o genitor e o segundo parâmetro o filho(a).



Depois de construída a base de conhecimento composta pelos fatos descritos acima, construa regras para representar as seguintes relações **binárias** de parentesco. Sempre o primeiro parâmetro deve ser considerado como na relação genitor descrita anteriormente. Assim, pai(X,Y) indica que X é pai de Y.

- *Esposo*
- *Esposa*
- *Mãe*
- *Pai*
- *Filha*
- *Filho*
- *Irmão*
- *Irmã*
- *Avô*
- *Avó*
- *Tia*
- *Tio*
- *Primo*
- *Prima*
- *Cunhado*
- *Cunhada*
- *Neto*
- *Neta*
- *Descendente* (Esta regra é recursiva. Pesquise sobre como fazê-la).

Escreva as seguintes consultas utilizando a sintaxe do Prolog e descreva o resultado da execução:

b) Quem são os netos e netas de Elizabeth?
c) Quem são os cunhados de Diana?
d) Quem são os bisavós de Zara?
e) Por que alguns resultados de consultas aparecem duplicados? Isto pode ocorrer, por exemplo, no caso da regra *primo*.

DICA IMPORTANTE: Definir relações simétricas em Prolog, usando a forma a seguir pode ser problemático:

```
conjuge(X,Y) :- conjuge(Y,X).
```

O problema é que o algoritmo de inferência do Prolog pode se perder resolvendo as definições recursivas infinitamente, pois não foi definido um critério de parada (Insira esta linha no seu código e repita a busca da questão 3b para verificar). Caso você se depare com uma situação onde uma declaração simétrica deste tipo é necessária, o melhor é criar uma definição auxiliar, esta sim simétrica, como mostrado a seguir:

```
casados(X,Y) :- conjuge(X,Y);conjuge(Y,X).
```

Utilize a relação auxiliar no restante do seu código, em vez de utilizar a relação original.

BOM TRABALHO!