Lógica Computacional

Caderno de Exercícios

Maria João Frade
Departamento de Informática
Universidade do Minho
2006

2^o Ano LMCC (2005/06)

Conteúdo

1 Exercícios Teórico-Práticos

1.1 Ficha 1

1. Converta as seguintes fórmulas de $\mathcal{L}_{\mathcal{P}}$ em fórmulas de $\mathcal{H}_{\mathcal{P}}$, $\mathcal{L}_{\mathcal{P}n}$ e $\mathcal{S}_{\mathcal{P}}$.
   (a) $p \wedge r$
   (b) $\neg(p \wedge q)$
   (c) $p \rightarrow q; (r \wedge s)$
2. Converta a fórmula $(((a \supset b) \wedge c) \supset a) \vee \neg c$ de $\mathcal{L}_{\mathcal{P}}$ num *grafo de Shannon* ($\mathcal{S}_{\mathcal{P}}$).
3. Considere os seguintes modelos de $\mathcal{L}_{\mathcal{P}}$:
   $$M_1 = \{p, q\} \quad \text{e} \quad M_2 = \{q\}$$
   Determine a validade de cada uma das proposições seguintes no modelo M_1 e no modelo M_2:
   (a) $p \supset (q \wedge p)$
   (b) $(p \vee q) \supset \neg q$
4. Para cada uma das proposições seguintes apresente (se possível) um modelo que a valide e um que a refute.
   (a) $p \supset r$
   (b) $p \wedge r$
   (c) $\neg p \wedge \neg r$
   (d) $\neg(p \wedge r)$
   (e) $p \wedge \neg p$
   (f) $p \vee (p \supset r)$
5. Quais das seguintes consequências semânticas se verificam? Justifique.
   (a) $\{p \supset r\} \models p \wedge r$
   (b) $\{p, r\} \models p \supset r$
   (c) $\models p \vee \neg p$
   (d) $\{p \wedge \neg p\} \models p \vee r$
6. Verifique que $((P \supset Q) \supset P) \supset P$ é uma tautologia.

1.2 Ficha 2

1. Determine a *forma normal negativa* equivalente a cada uma das seguintes proposições:
   (a) $(p \vee t) \supset (p \wedge t)$
   (b) $(p \wedge (p \supset t)) \supset \neg p$
   (c) $((p \supset t) \supset p) \supset p$
   (d) $(\neg p \supset t) \supset (\neg p \supset \neg t) \supset p$
2. Construa a *forma normal conjuntiva* e a *forma normal disjuntiva* equivalente a cada uma das proposições da alínea anterior.
3. Usando o *algoritmo de resolução de Robinson*, verifique se os seguintes conjuntos de cláusulas (na FNC) são inconsistentes:
   (a) $\{\{a, b, \neg c\}, \{\neg a, c\}, \{\neg b, e\}\}$
   (b) $\{\{\neg a\}, \{a, \neg b\}, \{a, \neg c\}, \{b, c\}, \{\neg b\}\}$
4. Usando o *algoritmo de resolução de Robinson*, verifique se os seguintes conjuntos de cláusulas (na FND) são tautologias:
   (a) $\{\{a, \neg b\}, \{a, b\}, \{\neg a\}\}$
   (b) $\{\{a, \neg b\}, \{a, b\}, \{\neg c, \neg a\}\}$

1.3 Ficha 3

1. Relembre o *algoritmo de Davis Putnam* para testar se uma FNC é inconsistente (trabalhando com FNC representadas como conjuntos de conjuntos de literais), e tenha em consideração os aspectos de simplificação do algoritmo. Use este algoritmo para verificar se:

   (a) $\{\{a, \neg b\}, \{\neg a, \neg c, d\}, \{a, \neg d\}, \{\neg a, c, b\}, \{c\}\}$ é inconsistente.

   (b) $\{a \supset b \wedge c, b \supset \neg d, a \wedge (\neg b \vee \neg e \vee d), c \supset e\}$ é inconsistente.

   (c) $\{p \supset s \wedge r, \neg q \vee s, \} \models p \wedge q \supset r$

2. O seguinte programa Prolog constroi a árvore de fraccionamento de Davis Putnam pelo próprio processo de prova.

```
inconsistente([]) :- fail.
inconsistente(P) :- member([],P).
inconsistente(P) :- fracciona(P,P1,P2),
                    inconsistente(P1), inconsistente(P2).
```

   (a) Defina o predicado `fracciona/3`.

   (b) Indique o que teria que fazer para contemplar os aspectos de simplificação do algoritmo.

3. Seja $\mathcal{P} = \{a, b, c\}$ o conjunto de símbolos proposicionais de $\mathcal{L}_\mathcal{P}$. Recorde a função de representação canónica $\Omega : \mathcal{L}_\mathcal{P} \longrightarrow \mathcal{P}(\mathcal{M}_\mathcal{P})$ que associa a cada fórmula o conjunto dos modelos que a validam (e que é um isomorfismo de álgebras booleanas). Indique o valor de

   (a) $\Omega(a)$

   (b) $\Omega(\neg a)$

   (c) $\Omega(b)$

   (d) $\Omega(b \wedge \neg a)$

   (e) $\Omega(a \vee b)$

   (f) $\Omega(\neg(a \vee b))$

4. Seja $\mathcal{P} = \{p, q, r, s, u\}$ o conjunto de símbolos proposicionais de $\mathcal{L}_\mathcal{P}$. Construa uma fórmula Φ que tenha como representação o conjunto de modelos Θ (ou seja, $\Omega(\Phi) = \Theta$), quando

   (a) $\Theta = \{\{u, p\}, \{r, u\}, \{r\}\}$

   (b) $\Theta = \{\{q, p\}, \{r, s, u\}, \{p, s\}\}$

   (c) $\Theta = \{\{s, p\}, \{\}, \{r\}\}$

1.4 Ficha 4

1. Assuma um conjunto finito de símbolos proposicionais definido pelo alfabeto em letras minúsculas, e a seguinte ordem total nesse conjunto de símbolos: $a > b > c > d > ... > z$.

   Construa o *Diagrama de Decisão Binária* (DDB) tipado que representa cada uma das seguintes fórmulas.

   (a) $b \wedge c$

   (b) $a \supset a$

   (c) $(b \wedge c) \supset (a \vee b)$

   (d) $((b \wedge c) \wedge \neg b) \supset a$

   (e) $((b \wedge c) \supset (a \vee b)) \wedge (\neg c \wedge (d \supset a))$

   (f) $((a \supset b) \vee c) \wedge (b \vee \neg(a \wedge c))$

   (g) $(a \wedge (b \supset \neg(a \vee c))) \vee (a \supset c)$

2. Construa a fórmula representada por cada Diagrama de Decisão Binária obtido nas diferentes alíneas da questão anterior.

   Como relaciona a fórmula obtida com a respectiva fórmula da questão anterior que lhe deu origem ?

1.5 Ficha 5

Relembre o *algoritmo de resolução* para a *Linguagem de Cláusulas em Lógica de Primeira Ordem.*

1. Considere o seguinte conjunto de cláusulas $\Gamma = \{\ \neg p \vee \neg q \vee r,\ p \vee \neg q,\ q\ \}$. Use a resolução para demonstrar que
   (a) Γ não é inconsistente.
   (b) $\Gamma \vdash r$
   (c) $\Gamma \nvdash w$
2. Use a resolução para mostrar que o seguinte conjunto de cláusulas é inconsistente
$$\{\ \forall x.\neg p(x),\ \forall y.p(y) \vee \neg q(y),\ q(a)\ \}$$
3. Use a resolução para validar a validade da fórmula $\phi \supset \psi \supset \phi$.
4. Use a resolução para mostrar que da assunção $\forall x.\neg gosta(x, ana) \supset gosta(ana, x)$ se pode provar que $gosta(ana, ana)$.
5. Considere o seguinte conjunto de cláusulas na forma condicional (regras):
$$member(X, [X|R])$$
$$member(X, [H|T]) \Leftarrow member(X, T)$$
   (a) Converta cada uma destas fórmulas, em cláusulas na forma disjuntiva.
   (b) Usando a resolução, demonstre que deste conjunto de cláusulas se infere $member(X, [a, b])$, indicando explicitamente as unificações necessárias. Ou seja, demontre que

$$\{\ \forall X, R.member(X, [X|R]),\ \forall X, H, T.member(X, [H|T]) \Leftarrow member(X, T)\ \} \vdash\ \exists X.member(X, [a, b])$$

6. Considere o seguinte conjunto de cláusulas na forma condicional (regras):
$$soma(0, X, X)$$
$$soma(s(X), Y, s(Z)) \Leftarrow soma(X, Y, Z)$$
$$mult(0, X, 0)$$
$$mult(s(X), Y, Z) \Leftarrow mult(X, Y, W) \wedge soma(W, Y, Z)$$
   (a) Converta cada uma destas fórmulas, em cláusulas na forma disjuntiva.
   (b) Usando a resolução, prove que deste conjunto de cláusulas se infere (indique explicitamente as unificações necessárias):
      i. $soma(s(s(0)), s(X), Y)$
      ii. $mult(s(0), s(s(0)), M)$

2 Exercícios Práticos - Prolog

2.1 Grupo I

1. Considere o seguinte programa Prolog:

```
% pertence(X,L) indica que X é um elemento da lista L
pertence(X,[X|_]).
pertence(X,[_|T]) :- pertence(X,T).

% prefixo(L1,L2) indica que a lista L1 é prefixo da lista L2
prefixo([],_).
prefixo([X|Xs],[X|Ys]) :- prefixo(Xs,Ys).

% sufixo(L1,L2) indica que a lista L1 é sufixo da lista L2
sufixo(L,L).
sufixo(Xs,[Y|Ys]) :- sufixo(Xs,Ys).

% concatena(L1,L2,L3) indica qua a lista L1 concatenada com a lista L2
% é a lista L3
concatena([],L,L).
concatena([H|T],L1,[H|L2]) :- concatena(T,L1,L2).
```

   Carregue estas definições no interpretador e interrogue a base de conhecimento. Por exemplo:

```
? pertence(2,[1,2,3]).
? pertence(X,[1,2,3]).
? pertence(3,L).
? pertence(X,L).
```

2. Considere a representação dos números naturais baseada nos construtores 0 e sucessor: 0, $suc(0)$, $suc(suc(0))$, $suc(suc(suc(0)))$, ...

   O predicado nat que testa se um termo é um número natural.

```
nat(0).
nat(suc(X)) :- nat(X).
```

   Defina, usando o functor `suc`, predicados que implementem as seguintes relações:

   (a) menor ou igual
   (b) mínimo
   (c) soma
   (d) multiplicação
   (e) factorial
   (f) exponenciação

3. Defina o predicado `last(X,L)` que testa se `X` é o último elemento da lista `L`.

4. Defina a relação `divide(L,L1,L2)` que divide a lista `L` em duas listas `L1` e `L2` com aproximadamente o mesmo tamanho.

2.2 Grupo II

1. Defina os seguintes predicados sobre listas:
   (a) `minimo/2` que produz o menor elemento presente numa lista.
   (b) `somatorio/2` que cálcula o somatório de uma lista.
   (c) `nesimo/3` que dá o elemento da lista na n-ésima posição
2. Defina um procedimento que ordene de uma lista segundo o algoritmo *quicksort*.
3. Considere o seguinte procedimento para o cálculo do factorial

   ```
   fact(0,1).
   fact(N,F) :- N>0, N1 is N-1, fact(N1,F1), F is N*F1.
   ```

   Defina uma outra versão de factorial que utilize um parâmetro de acumulação.
4. Defina uma versão do predicado somatório que utilize um acumulador.

2.3 Grupo III

1. Defina a relação `flatten/2` (que lineariza uma lista) de forma a que, por exemplo:

   ```
   | ?- flatten([a,b,[c,d],[],[[[e,f]],g],h],X).
   X = [a,b,c,d,e,f,g,h] ?
   yes
   ```

2. Escreva um programa para reconhecer se uma fórmula da lógica proposicional está na forma normal conjuntiva, ou seja, é uma conjunção de disjunções de literais. Um literal é um símbolo proposicional ou a sua negação.

   Considere a declaração das seguintes conectivas lógicas:

   ```
   :- op(500,yfx,/\).
   :- op(500,yfx,\/).
   :- op(300,fy,~).
   ```

3. Defina o predicado `conta_ocorr/3` para contar quantas vezes uma constante ocorre numa lista. (Sugestão: usar `atomic/1`).
4. Suponha que tem factos da forma `quadrado(Lado)`. Defina o predicado `zoom(+X,?Y,+F)` tal que `Y` é o quadrado que resulta de multiplicar pelo factor `F` os lados do quadrado `X`. (Sugestão: usar `=..`).

2.4 Grupo IV

1. Considere os seguintes predicados Prolog:

```
% testa um termo representa uma árvore binária válida
arv_bin(vazia).
arv_bin(nodo(X,Esq,Dir)) :- arv_bin(Esq), arv_bin(Dir).

% verifica se um termo pertencea uma árvore binária
na_arv(X,nodo(X,_,_)).
na_arv(X,nodo(Y,Esq,_)) :- na_arv(X,Esq).
na_arv(X,nodo(Y,_,Dir)) :- na_arv(X,Dir).
```

2. Defina predicados que permitam fazer as travessias *preorder*, *inorder* e *postorder*.

3. Defina um predicado `search_tree/1` que teste se uma dada árvore é uma árvore binária de procura.

4. Defina a relação `insert_tree(+X,+T1,?T2)` que sucede se `T2` é uma árvore binária de procura resultado da inserção de `X` na árvore binária de procura `T1`.

5. Defina a relação `path(+X,+Tree,?Path)` que sucede se `Path` é o caminho da raiz da árvore binária de procura `Tree` até `X`.

6. Defina um predicado `no_dupl/2` que remova os duplicados de uma lista.

7. Defina um predicado `enumerar(+N,+M,+P,?L)` que gera a lista `L` de números entre `N` e `M`, a passo `P`. Por exemplo:

```
| ?- enumerar(3,10,2,L).
L = [3,5,7,9] ?
yes
```

8. Defina um programa que faça a ordenação de uma lista pelo algoritmo *merge sort*. Use o *cut* para implementar o predicado merge de forma mais eficiente.

2.5 Grupo V

1. Defina o predicado `subconj(-S,+C)` onde `S` e `C` são duas listas que representam dois conjuntos. Este predicado deve gerar, por backtracking, todos os subconjuntos possíveis de `C`.

2. Defina o predicado `partes(+C,-P)`, que dado um conjunto `C` (implementado como lista) dá em `P` o conjunto de todos os subconjuntos de `C`.

3. Considere a linguagem proposicional gerada pelos simbolos proposicionais (átomos) e as conectivas: `falso, verdade, ~, /\, \/, =>`. Relembre que um modelo é um conjunto de simbolos proposicionais.

   ```
   :- op(600,xfy,=>).
   :- op(500,yfx,/\).
   :- op(500,yfx,\/).
   :- op(300,fy,~).
   ```

   Defina o predicado `atrib(+M,+P,?V)` que sucede se `V` é o valor de verdade da proposição `P` no modelo `M`. (Relembre a função μ da aulas teóricas). Pode utilizar o predicado `simp(+E,?V)` que faz o cálculo do valor de uma expressão na álgebra booleana Z_2.

   ```
   simp(X+X,0).
   simp(X*X,X).
   simp(X+0,X).
   simp(0+X,X).
   simp(X*0,0).
   simp(0*X,0).

   simp(X+Y,Z) :- simp(X,X1), simp(Y,Y1), simp(X1+Y1,Z).
   simp(X*Y,Z) :- simp(X,X1), simp(Y,Y1), simp(X1*Y1,Z).
   ```

4. Defina os predicados `formula_valida(+M,+P)` e `teoria_valida(+M,+T)` que sucedem se a proposição `P` e teoria `T` é válida no modelo `M`.

   ```
   | ?- formula_valida([p,q], p/\q => q\/r => ~r /\ ~ ~p).
   yes
   ```

5. Defina o predicado `consequencia(+T,+P)` sucede se a proposição P é uma consequência (semântica) da teoria `T`. (Sugestão: gere primeiro todos os modelos possíveis com os simbolos proposicionais das fórmulas envolvidas.)

6. Defina `tautologia/1` que testa se uma fórmula é uma tautologia.

7. Defina `inconsistente/1` que testa se uma teoria é inconsistente.

8. Defina os predicados `soma/3` e `produto/3` que implementam as operações de soma e produto de conjuntos de conjuntos de literais (ver apontamento da aulas teóricas).

9. Defina o predicado `fnn(+P,-FNN)` que dada uma proposição (da linguagem da alínea 3) gera a forma normal negativa que lhe é semanticamente equivalente.

10. Defina o predicado `fnc(+FNN,-L)` que dada uma forma normal negativa, produz em `L` a forma normal conjuntiva equivalente, representada por conjuntos de conjuntos de literais.

11. Defina o predicado `constroi_fnc(+L,-P)` que sucede se `P` é a proposição que o conjunto de conjuntos de literarais `L` representa.

12. Defina o predicado `gera_fnc(+P,-FNC)` que dada uma proposição `P` produz `FNC` uma proposição na forma normal conjuntiva, semanticamente equivalente a `P`.

2.6 Grupo VI

1. Use a traçagem para confirmar a construção das árvores de procura que foram apresentas ao longo dos slides das aulas práticas.

2. A seguinte definição pretende contar o número de ocorrências de um elemento numa lista, usando um parâmetro de acumulação.

```
conta_errado(X,L,N) :- contaAC(X,L,0,N).

contaAC(_,[],Ac,Ac).
contaAC(X,[H|T],Ac,N) :- X==H, Ac1 is Ac+1, contaAC(X,T,Ac1,N).
contaAC(X,[_|T],Ac,N) :- contaAC(X,T,Ac,N).
```

   Mas este predicado não está correctamente definido. Por exemplo:

```
| ?- conta_errado(3,[3,2,3,4],N).
N = 2 ? ;
N = 1 ? ;
N = 1 ? ;
N = 0 ? ;
no
```

   Faça *debugging* deste predicado para detectar o erro, e corrija-o.

2.7 Grupo VII

1. Assuma que a informação referente aos horários das salas de aula está guardada na base de conhecimento em factos da forma `sala`(*num,dia,inicio,fim,discipl,tipo*). Por exemplo:

```
:- dynamic sala/6.

sala(cp1103, seg, 10, 13, aaa, p).
sala(cp2301, ter, 10, 11, aaa, t).
sala(di011, sab, 12, 10, xxx, p).  % com erro
sala(cp3204, dom, 8, 10, zzz, p).
sala(di011, sex, 14, 16, xxx, p).
sala(cp204, sab, 15, 17, zzz, tp).
sala(di011, qui, 14, 13, bbb, tp). % com erro
sala(di104, qui, 9, 10, aaa, tp).
sala(dia1, dom, 14, 16, bbb, t).
sala(cp1220, sab, 14, 18, sss, p).
```

   (a) O seguinte predicado, permite retirar da base de dados todas marcações de sala em que, erradamente, a hora de início da aula é superior à hora de fim.

```
apaga_erros :- sala(N,D,Hi,Hf,C,T), Hf =< Hi,
               retract(sala(N,D,Hi,Hf,C,T)), fail.
apaga_erros.
```

   Qual é o efeito da segunda clausula `apaga_erros` ?

   (b) Execute um programa que retire todas as marcações de salas para os domingos.

   (c) Execute um programa que retire todas as marcações de salas para os sábados depois das 13 horas.

   (d) Defina o predicado `ocupada(+NSala,+Dia,+Hora)` que sucede se a sala número `NSala` está ocupada no dia `Dia` à hora `Hora`.

   (e) Defina o predicado `livre(?NSala,+Dia,+Hora)` que sucede se a sala número `NSala` está livre no dia `Dia` à hora `Hora`.

   (f) Defina o predicado `livres(-Lista,+Dia,+Hora)` que sucede se `Lista` é a lista dos números das salas que estão livres no dia `Dia` à hora `Hora`.

   (g) Defina o predicado `total_ocupacao(+NSala,-Total)` que sucede se `Total` é o número total de horas que a sala `NSala` está ocupada.

   (h) Defina o predicado `cria_total(+NSala)` que acrescenta à base de dados um facto, associando à sala `NSala` o número total horas de ocupação dessa sala por semana.

   (i) Defina o predicado `intervalo_livre(?NSala,+Dia,+Inicio,+Fim)` que sucede se `NSala` está livre no dia `Dia` durante o periodo de tempo entre `Inicio` e `Fim`.

2. Assuma que a informação referente às notas dos alunos à disciplina de *Lógica Computacional 2005/2006* está guardada na base de conhecimento em factos da forma:

   `modalidadeA`(*numero, nome, fichas, exame*)
   `modalidadeB`(*numero, nome, fichas, trabalho, exame*)

   Por exemplo:

```
modalidadeA(1111, 'Maria Campos' , 14, 12).
modalidadeA(3333, 'Rui Silva', 13, 15).
modalidadeA(4444, 'Paulo Pontes', 17, 12).
modalidadeA(8888, 'Antonio Sousa', 14, 8).

modalidadeB(2222, 'Ana Miranda', 14, 15, 12).
modalidadeB(5555, 'Joao Ferreira', 15, 16, 11).
```

   (a) Escreva o predicado `gera_notas/0` que cria factos `nota/3`, que definem a relação entre o aluno (número e nome) e a sua nota final (calculada de acordo com o estabelecido para esta disciplina).

   (b) Defina o predicado `aprovados(-Lista)` que sucede se `Lista` contem o número dos alunos aprovados à disciplina.

3. Assuma que para implementar uma agenda electrónica temos na base de conhecimento factos com a seguinte informação `agenda`(*data, horaInicio, horaFim, tarefa, tipo*). Por exemplo:

```
agenda(data(5,abril,2006), 9, 13, join, palestras).
agenda(data(6,abril,2006), 11, 13, join, palestras).
agenda(data(6,abril,2006), 16, 17, logcomp, aulas).
agenda(data(6,abril,2006), 17, 20, atendimento, aulas).
agenda(data(4,abril,2006), 15, 17, di, reuniao).
agenda(data(7,abril,2006), 8, 13, logcomp, aulas).
agenda(data(7,abril,2006), 15, 17, ccc, reuniao).
agenda(data(4,maio,2006), 11, 13, pure, palestras).
```

   (a) Defina o predicado `cria_tipo(+Tipo)` que consulta a agenda e gera factos do nome do tipo com a lista de pares *(data,tarefa)* associados a esse tipo.

   (b) Defina o predicado `apaga_mes(+Mes)` que apaga todas as marcações de um dado mês.

   (c) Defina o predicado `marca(+Tarefa,+Tipo,+LDatas)` faz a marcação de uma dada tarefa, de um dado tipo, para uma lista de datas.

2.8 Grupo VIII

1. Defina o predicado `tabuada(+N)` que dado um número inteiro `N`, apresenta no écran a tabuada do `N`.

2. Escreva um programa `escreve_tabuadas` que lê um inteiro do teclado, escreve no écran a sua tabuada, e continua pronto para escrever tabuadas até que seja mandado terminar.

3. Escreva um programa que lê uma lista de pares *(átomo, n^o inteiro)* e apresenta um gráfico de barras dessa lista de pares. (Cada unidade deve ser representada pelo caracter #, e as barras podem ser horizontais.)

4. Escreva um programa que lê os coeficientes de um polinómio de 2^o grau ax^2+bx+c e calcula as raizes reais do polinómio, apresentando-as no écran. Se o polinómio não tiver raizes reais, o programa deve informar o utilizador desse facto.

5. Relembre o problema apresentado anteriormente, em que a informação referente aos horários das salas de aula está guardada na base de conhecimento em factos da forma:

   `sala`(*num,dia,inicio,fim,discipl,tipo*)

   Defina um predicado `salva(+Ficheiro)` que guarda no `Ficheiro` os factos com a informação sobre as salas que são válidas (i.e., em que a hora de início é inferior à hora de fim).

6. Defina o predicado `findterm(+Term)` que escreve o écran o primeiro termo lido que unifica com `Term`.

7. Defina o predicado `findalltermsfile(+Term,+FileName)` que escreve no écran todos os termos do ficheiro que unificam com `Term` (garanta que `Term` não é instanciado).

8. Defina o predicado `to_upper(+FileIn,+FileOut)` que recebe um ficheiro de texto `FileIn` e gera o ficheiro `FileOut` com o mesmo texto de entrada mas convertido para letras maiúsculas. (Note que apenas as letras minúsculas são alteradas, o resto deverá ser mantido.)

9. Defina o predicado `numera_linhas(+FileIn,+FileOut)` que recebe o ficheiro `FileIn` e produz o ficheiro `FileOut`, que contém as mesmas linhas de `FileIn`, mas com as linhas numeradas.

10. Relembre o problema do cálculo da nota final à disciplina de *Lógica Computacional*, apresentado anteriormente, em que as notas do alunos estão guardadas na base de conhecimento em factos da forma:

    `modalidadeA`(*numero, nome, fichas, exame*)
    `modalidadeB`(*numero, nome, fichas, trabalho, exame*)

    Pretende-se agora poduzir a pauta final, tendo a informação sobre os alunos inscritos num ficheiro com factos da forma: `aluno`(*numero, nome, tipo*)

    Defina o predicado `gera_pauta(+Inscritos,+Pauta)` que dado o ficheiro `Inscritos`, vai lendo a informação sobre os alunos inscritos à disciplina e, sem alterar a base de conhecimento, produz no ficheiro `Pauta` o texto com a pauta devidamente preenchida. Note que:

    - se o aluno está inscrito mas não se submeteu a avaliação, a sua nota será Faltou;
    - se o aluno tem alguma das componentes de avaliação inferior a 9, ou a média final arredondada inferior a 10, a sua nota será Reprovado;
    - nos restantes casos, será o arredondamento da média pesada (se quiser, pode escrever também a nota por extenso).