ECM253 – Linguagens Formais, Autômatos e Compiladores

P1 – 2020 – Atividades

Marco Furlan

16 de abril de 2020

Regras

(1) Esta **atividade** deverá ser **resolvida** em **equipe** – a mesma formada em sala de aula;

(2) **O que** deve ser **enviado**:

- Um **documento PDF** no link publicado no Moodlerooms da disciplina e contendo a solução dos problemas da atividade:
  - **Identificar** claramente no **início do documento** o **nome** e **RA** dos integrantes da equipes;
  - **Responder** os problemas **na ordem apresentada na atividade**, enumerando-os como nesta atividade;

(3) Esta atividade estará **disponível** a partir de **06/04/2020**;

(4) Sua solução deverá ser **enviada** até o dia **24/04/2020**.

§

(1) (**até 2,0 pontos**) Responder as questões a seguir sobre lógica de predicados.

(a) (**até 0,5 pontos**) Considerar um vetor A contendo apenas valores numéricos presente em um programa Python. Escrever as sentenças a seguir na forma simbólica:

(i) Todos os elementos de A são positivos.

(ii) Todos os elementos de A são positivos e menores ou iguais a 70.
(iii) Alguns dos valores são maiores que 60.
(iv) Os valores de A em qualquer linha estão ordenados de modo ascendente.
(v) Os valores de A em qualquer coluna estão ordenados de modo ascendente.
(vi) Os valores nas três primeiras linhas de A são distintos.
(vii) Os valores em quaisquer três linhas consecutivas de A são distintos.
(viii) Para quaisquer duas linhas consecutivas de A, a soma das entradas na segunda das linhas é 20 a mais do que a soma das entradas na primeira das linhas.

(b) (**até 0,5 pontos**) Provar a validade, a partir das hipóteses e utilizando regras de inferência e de equivalência, o argumento a seguir: $(\forall x)[p(x) \vee q(x)] \wedge (\forall x)[\neg p(x) \wedge q(x) \rightarrow r(x)] \rightarrow (\forall x)[\neg r(x) \rightarrow p(x)]$.

Considerar as seguintes leis adicionais da lógica proposicional para este exercício (também valem para lógica de predicados):

- Distributiva (dist): $(p \vee q) \wedge (p \vee r) \Longleftrightarrow p \vee (q \wedge r)$
- Identidade (ident): $p \vee \mathsf{F} \Longleftrightarrow p$ (F representa falso)
- Inversa (inv): $p \wedge \neg p \Longleftrightarrow \mathsf{F}$ (F representa falso)

(c) (**até 0,5 pontos**) Provar a validade ou determinar uma interpretação inválida para o argumento a seguir: $(\exists x)[P(x) \rightarrow Q(x)] \wedge (\forall y)[Q(y) \rightarrow R(y)] \wedge (\forall x)P(x) \rightarrow (\exists x)R(x)$.

(d) (**até 0,5 pontos**) Transformar os argumentos a seguir em uma fórmula bem formada da logica de predicados e depois prove, a partir das hipóteses e utilizando regras de inferência e de equivalência, a sua validade:

> Alguns elefantes tem medo de todos os ratos. Alguns ratos são pequenos. Portanto, existe algum elefante que tem medo de alguma coisa pequena.

Usar os símbolos $E(x)$ (x é um elefante), $M(x)$ (x é um rato), $A(x, y)$ (x tem medo de y) e $S(x)$ (x é pequeno). **Dica**: após fazer a formulação, se for utilizada a identidade $A \wedge B \wedge C \rightarrow D \rightarrow E \equiv A \wedge B \wedge C \wedge D \rightarrow E$, ela facilitará significativamente a prova (ela foi provada em aula!). Esta identidade vale para lógica proposicional ou de predicados.

(2) (**até 2,0 pontos**) Responder as questões a seguir sobre conjuntos.

(a) (**até 0,5 pontos**) Sejam $A, B \subseteq \mathscr{U}$. Provar com argumentações lógicas: $B - A \subseteq \bar{A}$. **Exemplo**, para provar que $B - \bar{A} = B \cap A$, pode-se proceder assim: se x pertence à $B - \bar{A}$ é porque $x \in B$ e $x \not\in \bar{A}$. Mas se $x \not\in \bar{A}$ é porque $x \in A$. Então, sendo que $x \in B$ e $x \in A$ isso é o mesmo que dizer que $x \in B \cap A$, que é o que se queria provar.

(b) (**até 0,5 pontos**) Sejam $A, B, C \subseteq \mathscr{U}$. Provar com argumentações lógicas ou utilizando diagrama de Venn: $(A \cap B) \cup C = A \cap (B \cup C)$ se e somente se $C \subseteq A$.

(c) (**até 0,5 pontos**) Uma faculdade possui 300 alunos em seu curso de Computação. Sabe-se que 180 alunos estudam Python, 120 estudam Java, 30 estudam Matlab, 12 estudam Python e Matlab, 18 estudam Java e Matlab, 12 estudam Python e Java e 6 estudam as três linguagens. Pergunta-se: Quantos alunos estudam exatamente duas linguagens? Dica: Utilizar o princípio da inclusão e exclusão.

(d) (**até 0,5 pontos**) Uma pesquisa realizada com 150 alunos de uma faculdade revelou que 83 deles possuem automóvel, 97 possuem bicicleta, 28 possuem motocicleta, 53 possuem automóvel e bicicleta, 14 possuem automóvel e motocicleta, 7 possuem bicicleta e motocicleta e 2 possuem os três. Responder:

(i) Quantos alunos possuem apenas bicicleta e nada mais?

(ii) Quantos alunos não possuem nenhum dos meios de transporte pesquisados?

Dica: Utilizar o princípio da inclusão e exclusão.

(3) (**até 2,0 pontos**) Responder as questões a seguir sobre relações e funções.

(a) (**até 0,5 pontos**) Responder as questões a seguir, justificando a resposta:

(i) Para que conjuntos $A, B \subseteq \mathscr{U}$ é verdade que $A \times B = B \times A$?

(ii) Para $A, B, C \subseteq \mathscr{U}$, provar que $A \times (B - C) = (A \times B) - (A \times C)$.

(iii) Sejam A, B dois conjuntos e que $|B| = 3$. Se existem 4096 relações de A para B, qual é o valor de $|A|$?

(iv) Sejam $A, B \subseteq \mathbb{R}^2$, onde $A = \{(x, y) \mid y = 2x + 1\}]$ e $B = \{(x, y) \mid y = 3x\}]$. Determinar $A \cap B$.

(b) (**até 0,5 pontos**) Seja $g : \mathbb{N} \to \mathbb{N}$ definida por $g(n) = 2n$. Se $A = \{1, 2, 3, 4\}$ e $f : A \to \mathbb{N}$ é dada por $\{(1, 2), (2, 3), (3, 5), (4, 7)\}$, determinar $g \circ f$.

(c) (**até 0,5 pontos**) Se $A\{1, 2, 3, 4\}$ exemplificar relações $\mathscr{R}$ sobre A que sejam:

(i) Reflexiva e simétrica, mas não transitiva.

(ii) Reflexiva e transitiva, mas não simétrica.

(iii) Simétrica e transitiva, mas não reflexiva.

(d) (**até 0,5 pontos**) Para $A = \{1, 2, 3, 4, 5\}$ e $\mathscr{R} = \{(1, 1), (1, 2), (2, 1), (2, 2), (3, 3), (4, 4), (4, 5), (5, 4), (5, 5), (6, 6)\}$. Pergunta-se:

(i) Explique, usando a definição de relação de equivalência, por que $\mathscr{R}$ é uma relação de equivalência.

(ii) Calcule as classes de equivalência [1], [2] e [3].

(iii) Que partição de A é induzida por $\mathscr{R}$?

(4) (**até 2,0 pontos**) Da teoria de **logica proposicional**, tem-se as seguintes definições de fórmula bem-formada (fbf):

- V e F são fbfs;
- Um símbolo proposicional $(A, B, \ldots, Z)$ é uma fbf;
- Se A é uma fbf, então $\neg A$ também é uma fbf;
- Se A e B são fbfs, então também são $A \wedge B$, $A \vee B$, $A \to B$, $A \leftrightarrow B$; Se A é uma fbf, então também é (A).

E a prioridade dos operadores relacionais está representada pela tabela a seguir:

Ordem	Operador
1	()
2	$\neg$
3	$\wedge, \vee$
4	$\rightarrow$
5	$\leftrightarrow$

A precedência decresce de cima para baixo na tabela. Todos os operadores são binários, com exceção da negação.

Pede-se: Elaborar em Prolog uma **base de dados** contendo **regras** para a **análise sintática** de **expressões** da **lógica proposicional**. **Não** é para **interpretar** a **validade** das expressões, **apenas** para **responder** se uma **fórmula é bem-formada** ou deixar que o Prolog aponte o erro na sua expressão.

Para isso, será necessário definir um conjunto de operadores lógicos, bem como suas precedências. Um operador em Prolog é assim definido:

```
:- op(Precedência, Tipo, Func).
```

Onde `Precedência` é um número inteiro entre 0 e 1200 que indica a precedência do operador. `Tipo` indica a associatividade do operador e é descrito pela tabela a seguir:

Notação de tipo	Significado
`xfx`	Operador infixo não-associativo
`xfy`	Operador infixo associativo à direita
`yfx`	Operador infixo associativo à esquerda
`fx`	Operador prefixo não associativo
`fy`	Operador prefixo associativo à direita
`xf`	Operador pós-fixo não associativo
`yf`	Operador pós-fixo associativo à esquerda

Por fim, `Func` representa o símbolo do operador a ser definido. Para resolver este problema, utilize os símbolos a seguir:

Operador matemático	Operador em Prolog	Precedência	Associatividade
()	Não precisa definir – utilizar a definição original	-	-
$\neg$	~	501	Prefixo associativo à direita
$\wedge, \vee$	&, \/	502	Infixo associativo à esquerda
$\rightarrow$	==>	503	Infixo associativo à esquerda
$\leftrightarrow$	<==>	504	Infixo associativo à esquerda

Exemplo de um operador:

```
:- op(501, fy, ~).
```

Para resolver o problema, depois de definir os operadores como indicado anteriormente, basta escrever regras para definir o que é uma fbf. **Sugestão**: defina o predicado unário `fbf` para cada uma das definições de fbf já apresentada. Utilize os operados nesta definição. Em Prolog, utilizar as palavras reservadas `true` e `false` do Prolog para representar respectivamente os valores verdadeiro e falso. Exemplo de uma regra:

```
fbf((X)):-
    fbf(X).
```

Que diz: "se `X` é uma fbf, então `(X)` também é".

Depois, testar a sua base de dados com fbfs corretas e incorretas. Por exemplo, a fbf a seguir é bem-formada:

```
5 ?- fbf((a \/ b) ==> ~c).
true .
```

Mas a fbf a seguir não é:

```
6 ?- fbf((a \/ b) ~ ==> c).
ERROR: [Thread pdt_console_client_0_Default Process] Syntax error:
       Operator expected
ERROR: [Thread pdt_console_client_0_Default Process] fbf((a \/ b)
ERROR: [Thread pdt_console_client_0_Default Process] ** here **
ERROR: [Thread pdt_console_client_0_Default Process]  ~ ==> c) .
```

Adicionar cinco exemplos de fbf corretas e cinco exemplos de fbf incorretas em comentários dentro do arquivo de solução desenvolvido.

(5) (**até 2,0 pontos**) Reconsiderar a base de dados Prolog que resolve o problema do "macaco e a banana":

```
% move(State1, Move, State2): Move in State1 results in State2
% state(HorizPosition,VertPosition, BoxPosition, HasBanana):
%    configures the current state of the problem

move(state(middle,onbox,middle,hasnot),
        grasp,
        state(middle,onbox,middle,has)).

move(state(P,onfloor,P,H),
        climb,
        state(P,onbox,P,H)).
```

```
move(state(P1,onfloor,P1,H),
        push(P1,P2),
        state(P2,onfloor,P2,H)).

move(state(P1,onfloor,B,H),
        walk(P1,P2),
        state(P2,onfloor,B,H)).

% canget( State ): indicates how the monkey can get the banana

canget(state(_,_,_,has)).

canget(State1):-
        move(State1,Move,State2),
        canget(State2).
```

Para resolver o problema, utilizar a consulta a seguir, por exemplo:

```
?- canget(state(atdoor,onfloor,atwindow,hasnot)).
```

Pede-se: Alterar a base de dados apresentada acima para apresentar na tela o planejamento de ações de modo que o macaco consiga pegar a banana. O resultado pode ser apresentado de modo reverso. **Exemplo** de solução:

```
        END
        ---------------
        From: state(middle,onbox,middle,hasnot)
        Action: grasp
        To: state(middle,onbox,middle,has)
        ---------------
        From: state(middle,onfloor,middle,hasnot)
        Action: climb
        To: state(middle,onbox,middle,hasnot)
        ---------------
        From: state(atwindow,onfloor,atwindow,hasnot)
        Action: push(atwindow,middle)
        To: state(middle,onfloor,middle,hasnot)
        ---------------
        From: state(atdoor,onfloor,atwindow,hasnot)
        Action: walk(atdoor,atwindow)
        To: state(atwindow,onfloor,atwindow,hasnot)
        true.
```

Utilizar os predicados `write` (escreve uma mensagem ou átomo na tela), `nl` (pula linha) e `!` (força o backtracking para explorar todas as soluções). Conferir (de baixo para cima) que a partir da posição inicial, e com as ações escolhidas, o problema é resolvido.