INSTITUTO MAUÁ DE TECNOLOGIA

MAUÁ

ECM253 – Linguagens Formais, Autômatos e Compiladores

Trabalho 01

Marco Furlan

Abril/2021

Instruções

- Esta **atividade** deverá ser **resolvida** em **equipe** – a mesma formada em sala de aula;
- **Responder** todas as **questões na ordem apresentada** em um arquivo Word ou LaTeX e depois **exportar** para **PDF**;
- **Identificar** claramente no **início do documento** o **nome** e **RA** dos integrantes da equipes;
- **Enviar o PDF** criado para o **link do trabalho** no **MyOpenLMS da disciplina**;
- Apenas um integrante da equipe precisa enviar;
- Esta atividade estará **disponível** a partir de **15/04/2020**;
- A solução deverá ser **enviada** até o dia **22/04/2020**.

(1) (**até 3,0 pontos**) Responder as questões a seguir sobre lógica de predicados.

(a) (**até 1,5 ponto**) Provar a validade, utilizando regras de inferência e de equivalência, do argumento a seguir:

$$(\forall x)[P(x) \vee Q(x)] \wedge (\forall x)[\neg P(x) \wedge Q(x) \rightarrow R(x)] \rightarrow (\forall x)[\neg R(x) \rightarrow P(x)]$$

(b) (**até 1,5 ponto**) **Transformar o argumento a seguir** em uma **fórmula bem formada da logica de predicados** e, **depois**, **provar 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**) Provar as seguintes identidades com conjuntos. **Exemplo de prova**. 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 $x \notin \bar{A}$. Mas se $x \notin \bar{A}$ 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.

(a) (**até 1,0 pontos**) $B - A \subseteq \bar{A}$.

(b) (**até 1,0 pontos**) $(A \cap B) \cup C = A \cap (B \cup C)$ se e somente se $C \subseteq A$.

(3) (**até 3,0 pontos**) Utilizar o **princípio da inclusão e exclusão** para resolver os problemas a seguir.

(a) (**até 1,5 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?

(b) (**até 1,5 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?

(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 $(A, B, ..., Z)$ é uma fbf;
- Se A é uma fbf, então $\neg A$ também é uma fbf;
- Se A e B são fbfs, então também são $A \wedge B$, $A \vee B$, $A \rightarrow B$, $A \leftrightarrow B$; 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	¬
3	∧, ∨
4	→
5	↔

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

A linguagem **Prolog** permite a **redefinição e criação de operadores**. Um **operador** em Prolog é assim **criado**:

```
:- op(prec, tipo, func).
```

Onde `op` é um **predicado Prolog** que permite **criar um operador**, `prec` é um **número inteiro entre 0 e 1200** que indica a **precedência** do operador (quanto menor o valor, maior será a precedência) e `tipo` indica a **associatividade do operador**, especificado 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` é o **símbolo do operador** (pode ter vários caracteres de extensão). **Exemplo** da criação de um operador em Prolog:

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

Neste exemplo, o nome do operador é "`~`", sua precedência é 501 e sua associatividade é "`fy`", que representa "operador prefixo associativo à direita". Ou seja este operador poderá ser utilizado em expressões como "`~X`".

Pode-se utilizar este operador para definir que a negação de uma fórmula bem formada (`fbf`) também é uma fórmula bem formada (teste no Prolog):

```
% definição de operadores
:- op(501, fy, ~).
% símbolos constantes são fbf's
fbf(a).
% regras
% se X é uma fbf, então ~X é uma fbf
fbf(~X):-
    fbf(X).
```

Então, a consulta a seguir é verdadeira:

```
4 ?- fbf(~a).
true.
```

Tarefa:

- Elaborar em Prolog uma **base de dados** contendo **regras** que permita **verificar** se **expressões** são **fbfs** da **lógica proposicional**;
- **Não** é para **interpretar** as expressões – deve-se **apenas verificar** se uma **fórmula é bem-formada** ou **não**;
- Para tanto, **definir** em **Prolog** um **conjunto de operadores lógicos**, bem como suas **precedências**, conforme descrito na tabela a seguir (notar que os operadores em Prolog foram adaptados para representar os mesmos da Lógica Proposicional):

Operador matemático	Operador em Prolog	Precedência	Associatividade
`()`	Não precisa definir – utilizar a definição original	-	-
¬	`~`	501	Prefixo associativo à direita
∧, ∨	`&`, `\/`	502	Infixo associativo à esquerda
→	`==>`	503	Infixo associativo à esquerda
↔	`<==>`	504	Infixo associativo à esquerda

Então, para **resolver o problema**, crie uma base de dados Prolog seguindo a **sequência apresentada** a seguir:

- Definir os operadores;
- Uma fbf pode ser `true` e `false`;
- Definir fbf's básicas – literais – letras constantes de "`a`" à "`z`";
- Definir regras para especificar fbfs válidas, conforme apresentado anteriormente.

Por fim, testar a base de dados criada 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) .
```

Na base de dados desenvolvida, **adicionar cinco exemplos** de **fbf corretas** e **cinco exemplos** de **fbf incorretas em comentários** dentro do arquivo de solução desenvolvido.