ATIVIDADES AVALIATIVAS – HASKELL

Este trabalho pode ser realizado em grupo de até 4 alunos. Trabalhos de grupos com mais de 4 alunos serão zerados. Para que seu trabalho seja avaliado você deverá postar, no ambiente virtual de aprendizagem, na área reservada para este fim, um link para um ambiente online (https://www.onlinegdb.com/) contendo a solução de cada uma das atividades avaliativas que serão postadas no Ambiente Virtual de Aprendizado. Todos os códigos enviados devem conter o enunciado do trabalho que está sendo resolvido, na forma de comentário, precedido pelo nome dos integrantes do grupo em ordem alfabética.

Você deve fazer cada atividade avaliativa em um arquivo próprio, específico e único. Todas as atividades avaliativas deverão ser resolvidas em Haskell. Esta avaliação pode incluir 1, 2 ou 3 tarefas diferentes.

Lembre-se que nenhum trabalho, exercício, pesquisa científica, ou acadêmica, admite plágio. Todos os conceitos que você trouxer para o seu trabalho deverão ser inéditos, ou estar citados. Qualquer indicação de plágio, ou dúvida sobre autoria do trabalho, provocará, ou o zeramento do trabalho ou a defesa presencial, individual e sem consulta do trabalho apresentado. No caso de trabalhos em grupo o professor irá sortear o aluno que fará a defesa.

O conjunto das atividades avaliativas valem até 4 pontos na nota da RA2, ou seja, tem peso 4. ATENÇÃO, O PROFESSOR IRÁ SORTEAR UM ALUNO DO GRUPO PARA DEFENDER A TAREFA PESSOALMENTE E ESTA DEFESA IRÁ DETERMINAR A NOTA DO GRUPO. Vamos considerar 50% da tarefa, o atendimento de todas as condições aqui apresentadas e 50% a defesa feita pelo integrante sorteado.

Provedor de Lógica Proposicional com Geração de Cláusulas de Horn.

Tarefa: **Desenvolva um programa em Haskell que**:

1. **Leia expressões de lógica proposicional fornecidas pelo usuário, suportando**:
   - Conectivos lógicos: conjunção (∧ ou `and`), disjunção (∨ ou `or`), negação (¬ ou `not`), implicação (→ ou `=>`), bicondicional (↔ ou `<=>`). Ou a representação destes operadores em latex.
   - Parênteses para definir a precedência das operações.
   - Variáveis proposicionais representadas por letras maiúsculas (por exemplo, `A`, `B`, `C`).

2. **Avalie se a expressão é**:
   - Uma tautologia (verdadeira em todas as interpretações).
   - Uma contradição (falsa em todas as interpretações).
   - Contingente (verdadeira em algumas interpretações e falsa em outras).

3. **Converta a expressão proposicional em um conjunto equivalente de cláusulas de Horn**:
- Se for possível, exiba as cláusulas de Horn resultantes.
- Se não for possível, informe ao usuário que a expressão não pode ser representada apenas com cláusulas de Horn.

4. Exiba ao usuário o resultado, indicando qual é o caso e, se contingente, forneça pelo menos uma atribuição de valores de verdade que satisfaz a expressão e outra que a falsifica. Além, é claro, da cláusula de Horn, se possível, ou a razão da impossibilidade.
   - Todas as saídas devem ser legíveis e inelegíveis no terminal e, além disso, devem estar disponíveis em latex entre $$ e $$ de forma que possam ser renderizada em notação matemática em qualquer editor latex online como: Online LaTeX Equation Editor
5. Trate erros de sintaxe, informando ao usuário se a expressão fornecida é inválida.

Requisitos:

1. Implemente um analisador sintático (parser) manualmente, sem usar bibliotecas como Parsec, Megaparsec, ou qualquer outra biblioteca que faça o que foi solicitado nesta tarefa.
2. Utilize tipos de dados algébricos para representar a estrutura das expressões lógicas.
3. Implemente um algoritmo para:
   a. - Avaliar a expressão em todas as interpretações possíveis (por exemplo, gerando a tabela-verdade).
   b. - Converter a expressão em forma normal conjuntiva (CNF) e, em seguida, tentar representá-la como um conjunto de cláusulas de Horn.
4. Utilize monads, como a monad `Either`, para manejar possíveis erros durante o parsing e a avaliação.
5. Estruture o código de forma clara, separando as etapas de tokenização, parsing, avaliação, conversão para cláusulas de Horn e interação com o usuário.
6. Não utilize bibliotecas externas; todo o código deve ser escrito por você. Mas, você pode usar qualquer função, ou tipo disponível no **Prelude**.