# CTL MODEL CHECKING

Igor Quintal Mendes – 8622353

Marcello de Paula Ferreira Costa - 7960690

Lucas Tomazela - 8124271

## O PROGRAMA

O programa foi inteiramente desenvolvido na linguagem de programação Python, utilizando a versão Python 3.5.2. Os testes foram conduzidos no sistema operacional Linux, na distribuição Ubuntu (versão 16.04 LTS). O programa utiliza bibliotecas externas, como o GraphViz e tabulate.

## COMO EXECUTAR

Para executar o programa em ambiente Linux, o usuário deverá ter instalado alguns pacotes que foram utilizados no desenvolvimento, que estão listados no arquivo `requirements.txt`. Assim, aconselhamos o usuário a seguir os seguintes passos:

1. Fazer o download do *source* do projeto do GitHub:
    a. Acessando o projeto através do endereço e fazer o download do *.zip*:

       https://github.com/travoul/ctl-model-checking/archive/master.zip

    b. Realizando o clone do repositório através do comando:

```
$ git clone https://github.com/travoul/ctl-model-checking.git
```

2. Acessar o diretório do repositório
3. Instalar o virtualenv:

```
$ pip install virtualenv
```

4. Criar um ambiente virtual usando o virtualenv:

```
$ virtualenv venv -p python3
```

5. Ativar o ambiente criado com o virtualenv:

```
$ source venv/bin/activate
```

6. Instalar os pacotes indicados pelo arquivo `requirements.txt`:

```
$ pip install -r requeriments.txt
```

7. Para executar o ctl-model-checking:

```
$ python3 main.py <input.txt>
```

Neste caso, o arquivo `input.txt` contém a descrição da máquina de estados e também uma expressão CTL que será avaliada. Assim, caso não existam problemas na expressão CTL nem na descrição da máquina de estados, o programa deverá imprimir no terminal, ao final de sua execução, a árvore e o grafo gerados.

Visando facilitar a verificação de erros, bem como a visualização dos resultados, o usuário poderá executar a versão que utiliza o GraphViz. Assim, o usuário deverá ter o GraphViz instalado no seu computador, através do comando:

```
$ sudo apt install graphviz
```

Após a instalação do pacote, o usuário deverá acessar o diretório onde se encontra o *source* e executar no terminal:

```
$ python3 visual.py <input.txt>
```

Neste caso, o arquivo `input.txt` deverá seguir o mesmo formato como descrito anteriormente. Como resultado, serão gerados dois arquivos: `Graph.png` e `Tree.png`. O primeiro contém uma imagem do grafo que representa a máquina de estados e o segundo contém uma imagem da árvore sintática gerada pelo *parser* da expressão CTL.

## O ARQUIVO DE ENTRADA

O arquivo de entrada deve conter as informações da máquina de estado e uma expressão CTL a ser verificada. O formato seguido é o mesmo definido em aula, apresentado abaixo:

1. Primeira linha tem um valor totLinhas do tipo inteiro não nulo descrevendo o total de estados da máquina de estados
2. As próximas totLinhas linhas consecutivas terão os seguintes elementos separados por espaço
    - 2.1. Identificador único do estado (tipo inteiro)
    - 2.2. Total totProps de propriedades (tipo inteiro)
    - 2.3. Sequência de totProps strings representando os rótulos das propriedades dados como identificadores válidos, separadas por espaço
    - 2.4. Total de próximos totProxEstad estados (tipo inteiro)
    - 2.5. Sequência de totProxEstad identificadores válidos dos próximos estados, separados por espaço
3. Na última linha deverá constar a propriedade CTL a ser verificada.
    - 3.1. Verificar expressões com qualquer combinação válida de operadores;
    - 3.2. Considerar expressões contendo parênteses;
    - 3.3. Notificar propriedades mal-formadas;

## DEFININDO A CTL

A expressão CTL deve ser sempre **parentizada**. Cada termo deve aparecer entre parênteses, mesmo que seja unitário. Dois termos ou operadores nunca devem aparecer seguidos, sem uma devida parentização ("a|b", por exemplo).

Operadores também sempre requerem parênteses em seus operandos, e as funções lógicas & (*and*) e | (*or*) devem ser sempre separadas, mesmo que elas ocorram em conjunto. Por exemplo, "(a)|(b)|(c)" deve ser escrita como "((a)|(b))|(c)", ou "(a)|((b)|(c))". Note que também sempre é necessário adicionar parênteses na expressão por completo, de modo que para ser corretamente interpretado pelo programa, "(a)|((b)|(c))" deve ser inserido como "**(**(a)|((b)|(c))**)**".

Com relação aos operadores AX, EX, AF, EF, AG, EG, AU e EU, deve-se inserir os parênteses do operador mais os parênteses da expressão que ele deve avaliar. Como toda expressão é parentizada, se tivermos "EX(expressão)", e a expressão for apenas um único termo como "a", devemos ter como CTL "EX((a))". Neste caso, os parênteses em *verde* são relativos ao operador "EX", enquanto que os parênteses em ***vermelho*** são referentes a expressão "a".

Note também que a entrada pode ser fornecida com espaços entre os termos, contanto que a parentização correta seja respeitada.

Abaixo apresentamos uma série de exemplos de entradas válidas, justificando suas formatações:

- ( EG ( ( ( EX ( ( q ) ) ) & ( EX ( ( ! ( p ) ) ) ) ) ) )

Os parênteses em ***vermelho*** referem-se novamente a expressão contida no operador. Já os parênteses em *verde* referem-se aos dos operadores EX e EG. Note que os operadores lógicos "&" e "!" exigem também que a expressão a esquerda e a expressão a direita (no caso do "!", apenas a expressão da direita) estejam entre parênteses. Isto justifica os símbolos marcados em *azul*.

Porém, como toda a CTL é uma expressão, temos que ela também deve ser parentizada, e portanto, surgem os caracteres em *laranja*.

- ( AU ( ( a ) , ( b ) ) )

A mesma análise realizada para o exemplo anterior pode aqui também ser feita. O termo AU é composto de duas expressões, separadas por uma vírgula. Sua parentização é representada pelos parênteses em *verde*.

Como citado anteriormente, cada expressão dever ter sua própria parantização, e portanto, surgem os parênteses em ***vermelho*** para as expressões "a" e "b".

Os parênteses em *laranja* novamente representam a parentização da expressão CTL por completo.

As expressões abaixo seguem a mesma semântica:

- ( AX ( ( b ) ) )
- ( EF ( ( a ) ) )
- ( AG ( ( ( e ) -> ( AF ( ( ! ( e ) ) ) ) ) ) )

Note, que como citado anteriormente, um único termo constitui uma expressão, e portanto, deve ser parentizado. Desse modo, se tivermos apenas o termo "a" como CTL, devemos inserir "(a)" no arquivo de entrada, e não apenas "a" ou "((a))".

## EXEMPLOS DE EXPRESSÕES CTL TESTADAS

A seguir, temos algumas expressões CTL que foram testadas com o programa, o que poderá guiar o usuário a construir as expressões no modelo que possa ser interpretado pelo programa:

- (AU((AU((a),(b))),(c)))
- (AX((a)))
- (EX((a)))
- (AF((a)))
- (EF((a)))
- (AG((a)))
- (EG((a)))

- ((a)->(b))
- (AG(((a)->(b))))
- (EG(((EX((q)))&(EX((!(p)))))))
- (AF(((EX((q)))->(AX((q))))))
- (EU((p),((q)&(r))))
- (AU((!(p)),(!(q))))
- ((p)->(!(q)))
- (AU((!(h)),(c)))
- (AG(((s)->(AF((h))))))
- (AG(((e)->(AF((!(e)))))))
- ((a)<->(b))
- ((a)->((b)->(c)))
- ((a)<->((b)<->(c)))
- ((a)<->((b)->(c)))
- (AU((a),(EU((b),(c)))))
- (AU((AU((a),(b))),(AU((a),(b)))))
- (((a)->(b))<->((c)->(d)))
- (((a)<->(b))<->((c)<->(d)))
- ((AU((a),(b)))<->(AU((c),(d))))
- (AG(((s)->(AF((h))))))

## OS ARQUIVOS DO REPOSITÓRIO

Nesta seção, os arquivos contidos no repositório remoto *ctl-model-checking* foram listados e descritos brevemente com a finalidade de facilitar a localização de conteúdo dentro do repositório.

`requirements.txt` - Lista das dependências necessárias para a execução do *ctl-model-checking*.

`README.md` - Especificação de requisitos mínimos e necessários escritos em *markdown* para servir como página de capa do repositório remoto

`main.py` - Arquivo principal de execução que utiliza os módulos implementados dentro da pasta *ctl*.

`visual.py` - Arquivo de execução que somente gera imagens que descrevem a máquina de estados e uma expressão CTL fornecidas como *input.*

`ctl/models/graph.py` - Implementação dos modelos utilizados para representar uma máquina de estados. O arquivo contém a implementação de duas classes: *Graph* e *GraphNode,* que implementam um grafo.

`ctl/models/tree.py` - Implementação dos modelos utilizados para representar uma árvore sintática. O arquivo contém a implementação de duas classes: *Tree* e *TreeNode*, que implementam uma árvore sintática.

`ctl/utils/parser.py` - Implementação das ferramentas necessárias para tradução de uma Expressão CTL qualquer para uma equivalente que utilize o conjunto mínimo de operadores implementados pelo *ctl-model-checking*.

`ctl/utils/evaluator.py` - Implementação das ferramentas necessárias para a avaliação de uma Expressão CTL traduzida em cima de uma máquina de estados.