1. TUTORIAL SOFTWARE ENCADEAMENTO PROGRESSIVO E REGRESSIVO

O software em anexo traz uma implementação dos mecanismos de encadeamento progressivo e regressivo. A implementação foi desenvolvida na linguagem de programação Python pelo professor Edson E. Scalabrin da PUCPR e é uma recodificação parcial do código descrito por Bigus & Bigus, em 1993, em Python. Foram mantidos apenas os códigos necessários para montar e processar regras booleanas. Também não faz parte da recodificação da parte gráfica a implementação original. Fornecemos ao estudante um arquivo compactado denominado “ExpertSystem.zip”.

Conteúdo do pacote

Extraia o conteúdo do arquivo “ExpertSystem.zip”, você terá a seguinte estrutura de pastas e arquivos:

```
ExpertSystem
└── ExpertSystem
    ├── api
    │   ├── esBooleanRuleBase.py
    │   ├── esClause.py
    │   ├── esCondition.py
    │   ├── esEffector.py
    │   ├── esEffectorClause.py
    │   ├── esFact.py
    │   ├── esMenu.py
    │   ├── esRule.py
    │   ├── esRuleBase.py
    │   ├── esRuleVariable.py
    │   └──esVariable.py
    └── app
        └── cinema
```

```
├── Main.py
└── RuleBaseCinema.py
```

Para executar o software você deve ter o Python instalado e devidamente configurado. Se você ainda não possui essa ferramenta é sugerido que o estudante siga o passo a passo descrito nas páginas abaixo de acordo com o seu sistema operacional:

- Windows: http://python.org.br/instalacao-windows/
- MacOS: http://python.org.br/instalacao-mac/
- Linux: http://python.org.br/instalacao-linux/

Funcionamento do software

Esta seção apresenta o exemplo de encadeamento progressivo, o texto a seguir foi elaborado pelo prof. Edson E. Scalabrin, 2019, e segue na íntegra.

"Para ilustrar a aplicação do encadeamento progressivo sobre a base de regras "como escolher um meio de transporte para ir ao cinema", a partir daqui chamaremos tal base de "indo ao cinema". Os primeiros passos são:

- Carregar a base de regras "indo ao cinema" no mecanismo de inferência; para ajudar a experimentar o mecanismo de encadeamento progressivo, em anexo a esse material, poderá encontrar uma implementação Python. Algumas partes do código serão mostradas no desenrolar deste texto, em particular, aquelas que podem ser modificadas para compor outras bases de regras.

```
Regra 01:     SE    distancia > 5
          ENTÃO     deslocamento = carro
Regra 02:     SE    distancia > 1 E
                    tempo < 15
          ENTÃO deslocamento = carro
```

```
Regra 03:     SE   distancia > 1 E
              tempo > 15
         ENTÃO     deslocamento = a-pe
Regra 04:     SE   deslocamento = carro E
              localDoCinema = centro
         ENTÃO meioDeTransporte = taxi
Regra 05:     SE   deslocamento = carro E
              localDoCinema = bairro
         ENTÃO     meioDeTransporte = carro-proprio
Regra 06:     SE   deslocamento = a-pe E
              clima = ruim
         ENTÃO meioDeTransporte = a-pe-triste
Regra 07:     SE   deslocamento = a-pe E
              clima = bom
         ENTÃO     meioDeTransporte = a-pe-feliz
```

O conjunto de regras apresentadas são também exibidas em Python, onde a função *create* define o *script* que cria a base de regras "indo ao cinema". Esse código e as demais funcionalidades são fornecidas como anexo.

```
def create(self):
    distancia = RuleVariable(self.br, "distancia")
    distancia.set_labels("1 50")
    distancia.set_prompt_text("Qual é a distância até o cinema
[1,50]?")

    deslocamento = RuleVariable(self.br, "deslocamento")
    deslocamento.set_labels("carro a-pe")
    deslocamento.set_prompt_text(
        "Qual é a forma de deslocameto [carro, a-pe]?")

    tempo = RuleVariable(self.br, "tempo")
    tempo.set_labels("0 60")
    tempo.set_prompt_text("Qual é o tempo disponível [1,60]?")

    local_do_cinema = RuleVariable(self.br, "localDoCinema")
    local_do_cinema.set_labels("centro bairro")
    local_do_cinema.set_prompt_text("Onde é o cinema [centro,
bairro]]?")

    meio_de_transporte = RuleVariable(self.br, "meioDeTransporte")
    meio_de_transporte.set_labels(
        "taxi carro-proprio a-pe-triste a-pe-feliz")
```

```
    meio_de_transporte.set_prompt_text(
        "Qual é o meio de transporte [taxi, carro-proprio, a-pe-
triste, a-pe-feliz]]?")

    clima = RuleVariable(self.br, "clima")
    clima.set_labels("ruim bom")
    clima.set_prompt_text("Como está o clima [bom, ruim]?")

    c_equals = Condition("=")
    c_more_then = Condition(">")
    c_less_than = Condition("<")

    Regra01 = Rule(self.br, "Regra 01",
                   [Clause(distancia, c_more_then, "5")],
                   Clause(deslocamento, c_equals, "carro"))

    Regra02 = Rule(self.br, "Regra 02",
                   [Clause(distancia, c_more_then, "1"),
                    Clause(tempo, c_less_than, "15")],
                   Clause(deslocamento, c_equals, "carro"))

    Regra03 = Rule(self.br, "Regra 03",
                   [Clause(distancia, c_more_then, "1"),
                    Clause(tempo, c_more_then, "15")],
                   Clause(deslocamento, c_equals, "a-pe"))

    Regra04 = Rule(self.br, "Regra 04",
                   [Clause(deslocamento, c_equals, "carro"),
                    Clause(local_do_cinema, c_equals, "centro")],
                   Clause(meio_de_transporte, c_equals, "taxi"))

    Regra05 = Rule(self.br, "Regra 05",
                   [Clause(deslocamento, c_equals, "carro"),
                    Clause(local_do_cinema, c_equals, "bairro")],
                   Clause(meio_de_transporte, c_equals, "carro-
proprio"))

    Regra06 = Rule(self.br, "Regra 06",
                   [Clause(deslocamento, c_equals, "a-pe"),
                    Clause(clima, c_equals, "ruim")],
                   Clause(meio_de_transporte, c_equals, "a-pe-
triste"))

    Regra07 = Rule(self.br, "Regra 07",
                   [Clause(deslocamento, c_equals, "a-pe"),
                    Clause(local_do_cinema, c_equals, "bom")],
                   Clause(meio_de_transporte, c_equals, "a-pe-
```

```
feliz"))
    return self.br
```

- Definir um conjunto de valores iniciais para as variáveis na memória de trabalho.

MEMÓRIA DE TRABALHO
distancia = 2 deslocamento = *None* tempo = 10 localDoCinema = centro meioDeTransporte = *None* clima = ruim

Em *Python:*

```
self.br.set_variable_value("distancia", "2")
self.br.set_variable_value("deslocamento", None)
self.br.set_variable_value("tempo", "10")
self.br.set_variable_value("localDoCinema", "centro")
self.br.set_variable_value("meioDeTransporte", None)
self.br.set_variable_value("clima", "ruim")
```

- Examinar as cláusulas antecedentes de cada regra para determinar quais podem ser acionadas.

Como o valor para **distância** não é superior a cinco, a regra 01 não pode ser disparada, apenas uma regra é disparada, a regra 02. Em outras palavras, o conjunto de conflitos para o primeiro ciclo de correspondência encerra apenas uma regra, a regra 02.

```
Regra 01:     SE    distancia > 5
          ENTÃO     deslocamento = carro
Regra 02:     SE    distancia > 1 E
                    tempo < 15
          ENTÃO deslocamento = carro
```

A resolução de conflitos é simples: seleciona-se a regra única. Disparando o mecanismo de ação: a regra obriga a vincular o valor "carro" à variável "deslocamento" e adicioná-lo à memória de trabalho.

```
self.br.set_variable_value("deslocamento", "carro")
```

MEMÓRIA DE TRABALHO
distancia = 2 deslocamento = carro tempo = 10 localDoCinema = centro meioDeTransporte = *None* clima = ruim

Agora estamos prontos para o próximo ciclo de inferências. Comparamos as regras para determinar quais podem ser disparadas. Agora que **deslocamento** tem um valor, duas regras são candidatas – regra 04 e 05. A regra 05 requer *localDoCinema = bairro*, logo ela é falsa. Isso deixa apenas uma regra, a regra 04. Todas as cláusulas antecedentes são satisfeitas para regra 04, com *deslocamento = carro* ***e*** *localDoCinema = centro*.

```
Regra 04:     SE   deslocamento = carro E
                   localDoCinema = centro
              ENTÃO meioDeTransporte = taxi
Regra 05:     SE   deslocamento = carro E
                   localDoCinema = bairro
              ENTÃO     meioDeTransporte = carro-proprio
```

A aplicação da regra 04 resultado em novo valor para **meio de transporte** à memória de trabalho.

```
self.br.set_variable_value("meioDeTransporte", "taxi")
```

MEMÓRIA DE TRABALHO
distancia = 2 deslocamento = carro tempo = 10 localDoCinema = centro meioDeTransporte = taxi clima = ruim

O ciclo continua até que uma condição de parada seja atingida. Porém, percebe-se que apenas uma regra foi acionada novamente, a regra 04. O conjunto de conflitos está vazio, então a inferência progressiva encerra. Deve-se notar que se iniciou com quatro fatos e calculou-se dois outros fatos, derivando **taxi** como **meio de transporte**.

Mostramos aqui uma abordagem bastante simples de processamento de regras em modo progressivo, porém ela é não a mais eficiente para grandes bases de regras. Nesta linha, o algorítmo Rete constrói uma estrutura de dados de rede para gerenciar as dependências entre dados, testes de condição e regras. Esta abordagem minimiza o número de testes necessários para cada operação de correspondência (Forgy, 1982). Contudo, a grande parte dos sistemas que empregam encadeamento progressivo usam métodos que são menos eficientes, mas mais fáceis de implementar."

Edson E. Scalabrin (2019).

Executando o software

Para executar o software é necessário navegar até a pasta que contém o arquivo "Main.py" e executá-lo utilizando o Python. No Windows é possível realizar esse procedimento com o comando abaixo:

```
C:\Users\ExpertSystem\ExpertSystem\app\cinema>python
Main.py
```

A seguir o software apresenta uma tela com algumas opções:

```
             : [0] sair
       REGRAS: [1] carregar, [2] mostrar
 FORWARD-CHAIN: [3] carregar variaveis, [4] executar, [7] reset
BACKWARD-CHAIN: [5] carregar variaveis, [6] executar, [7] reset
Digite sua opcao:
```

Figura 1 – Menu de opções.

O primeiro passo é carregar as regras, para isso o estudante deve digitar o valor “1”.

O segundo passo é visualizar todas as regras carregadas, o estudante pode digitar o valor “2”. Em seguida o software apresentará a seguinte saída:

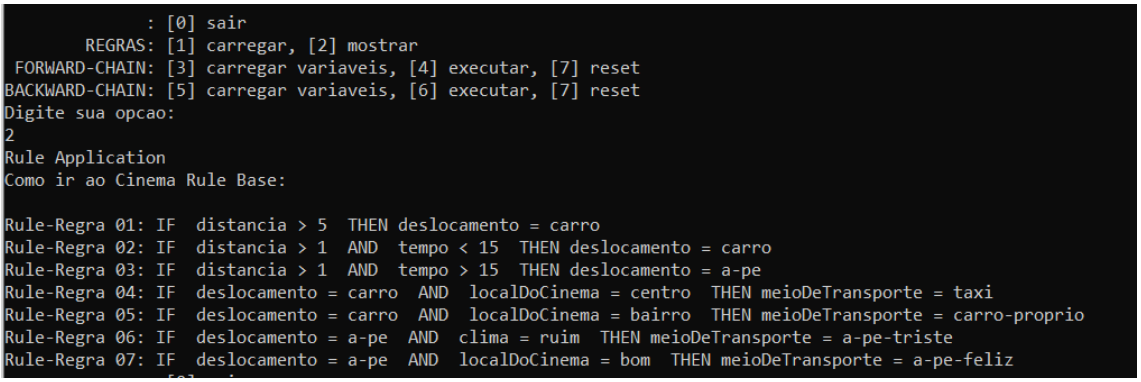


Figura 2 - Todas as regras carregadas.

O terceiro passo é carregar variáveis iniciais para o exemplo simples de encadeamento progressivo. Assim o estudante deve pressionar o valor “3”. Observe que ainda são desconhecidas as variáveis “deslocamento value” e “meioDeTransporte value”, sendo que ambas estão definidas como “None”.

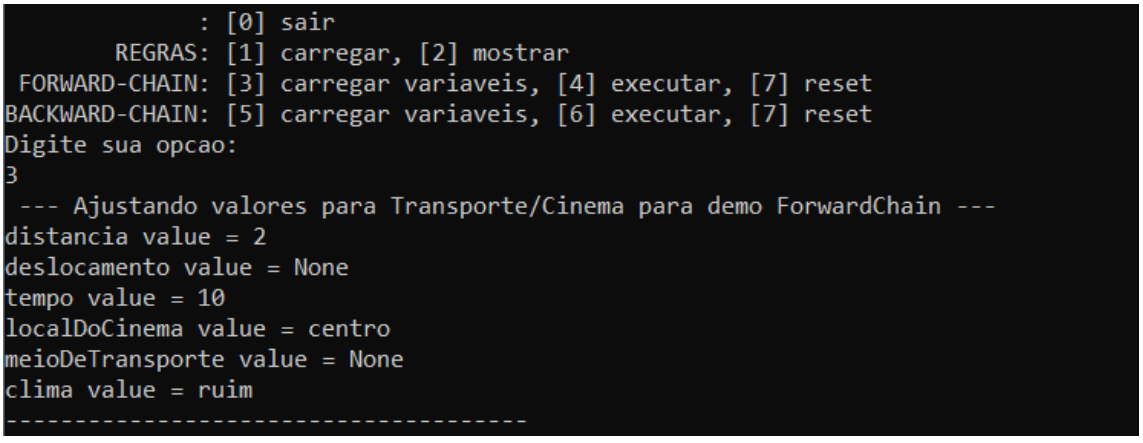


Figura 3 - Todas as variáveis iniciais carregadas para a demonstração do encadeamento progressivo.

O quarto passo é executar o encadeamento progressivo que vai disparar as regras com base nas variáveis (premissas) iniciais. O estudante pode digitar o valor “4”. Em seguida o software irá preencher as variáveis que estavam com valor None.

```
             : [0] sair
       REGRAS: [1] carregar, [2] mostrar
 FORWARD-CHAIN: [3] carregar variaveis, [4] executar, [7] reset
BACKWARD-CHAIN: [5] carregar variaveis, [6] executar, [7] reset
Digite sua opcao:
4
 --- Starting Inferencing Cycle ---
distancia value = 2
deslocamento value = carro
tempo value = 10
localDoCinema value = centro
meioDeTransporte value = taxi
clima value = ruim
--------------------------------------
```

Figura 4 – Estado final das variáveis após a execução do encadeamento progressivo.

O quinto passo é carregar as variáveis de encadeamento regressivo, para isso o estudante pode digitar o valor “5”. É possível ver que todas as variáveis iniciais (premissas) estão configuradas com o valor None, com exceção de “localDoCinema value” e “clima value”. Nesse caso onde temos poucas variáveis iniciais é interessante utilizar o encadeamento regressivo.

```
             : [0] sair
       REGRAS: [1] carregar, [2] mostrar
 FORWARD-CHAIN: [3] carregar variaveis, [4] executar, [7] reset
BACKWARD-CHAIN: [5] carregar variaveis, [6] executar, [7] reset
Digite sua opcao:
5
 --- Ajustando valores para Transporte/Cinema para demo BackwardChain ---
distancia value = None
deslocamento value = None
tempo value = None
localDoCinema value = centro
meioDeTransporte value = None
clima value = ruim
--------------------------------------
```

Figura 5 - Todas as variáveis iniciais carregadas para a demonstração do encadeamento regressivo.

O sexto passo é executar o encadeamento regressivo para isso o estudante pode digitar o valor “6”. Como observamos que existem poucas variáveis o software realiza o questionamento para o usuário para informar qual seria a variável objetivo, conforme Figura 6. O usuário deverá digitar as palavras “deslocamento” ou “meioDeTransporte”.

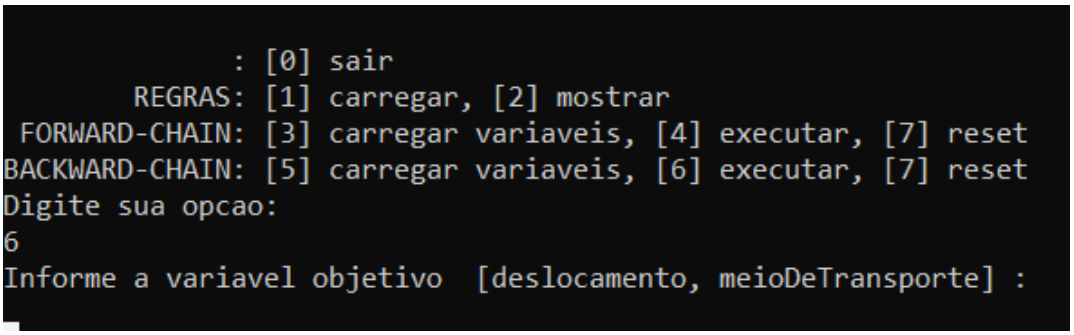
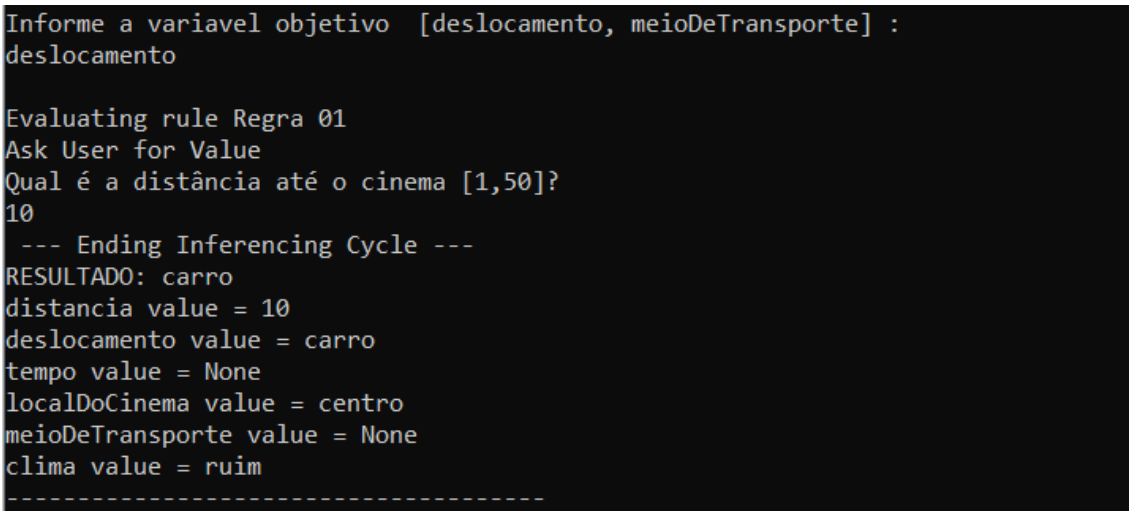


Figura 6 – Tela de iteração com o usuário para encontrar a variável objetivo.

Ao digitar “deslocamento” o software realizará outra interação com o usuário, solicitando uma entrada para a distância (entre “1” e “50”). Se o usuário digitar por exemplo o valor “10”, automaticamente o software irá executar o encadeamento regressivo com as variáveis definidas pelo usuário.

A Figura 7 apresenta o resultado do ciclo de inferência para encadeamento regressivo e mostra que a entrada “deslocamento value” foi configurada para “carro”. Chegando a um objetivo final.



Conteúdo dos arquivos

Os ajustes necessários para montar uma nova base de regras são feitos apenas nas classes Main e RuleBaseCinema. O arquivo “Main.py”, contém a classe Main e contém definições iniciais do software, apresentamos a seguir o conteúdo do arquivo “Main.py”:

```
from ExpertSystem.api.esMenu import APP
from ExpertSystem.app.cinema.RuleBaseCinema import RuleBaseCinema

class Main:
    def __init__(self):
        self.app = APP("Rule Application")

    def main(self):
        try:
            # RuleBaseCinema recebe dois parâmetros:
            # o primeiro é nome da base de regras e
            # o segundo é lista de várias presentes na base de regras.
            # Essas variáveis fazem parte dos possíveis objetivos
            brCinema = RuleBaseCinema("Como ir ao Cinema",
                                      "[deslocamento, meioDeTransporte] :")
            self.app.add_rule_base(brCinema)
            self.app.menu()
        except Exception as e:
            print("Exception: RuleApp ", e.with_traceback())


if __name__ == '__main__':
    Main().main()
```

Em especial a classe RuleBaseCinema implementa a maior parte das regras, ela está contida no arquivo “RuleBaseCinema.py”, cujo conteúdo é apresentado a seguir.

```
from ExpertSystem.api.esBooleanRuleBase import BooleanRuleBase
from ExpertSystem.api.esRuleVariable import RuleVariable
from ExpertSystem.api.esCondition import Condition
from ExpertSystem.api.esRule import Rule
from ExpertSystem.api.esClause import Clause
```

```
class RuleBaseCinema:
    def __init__(self, nome, goals_list):
        self.br = BooleanRuleBase(nome)
        self.goals_list = goals_list

    def get_goal_list(self):
        return self.goals_list

    def create(self):
        distancia = RuleVariable(self.br, "distancia")
        distancia.set_labels("1 50")
        distancia.set_prompt_text("Qual é a distância até o cinema [1,50]?")

        deslocamento = RuleVariable(self.br, "deslocamento")
        deslocamento.set_labels("carro a-pe")
        deslocamento.set_prompt_text(
            "Qual é a forma de deslocameto [carro, a-pe]?")

        tempo = RuleVariable(self.br, "tempo")
        tempo.set_labels("0 60")
        tempo.set_prompt_text("Qual é o tempo disponível [1,60]?")

        local_do_cinema = RuleVariable(self.br, "localDoCinema")
        local_do_cinema.set_labels("centro bairro")
        local_do_cinema.set_prompt_text("Onde é o cinema [centro, bairro]]?")

        meio_de_transporte = RuleVariable(self.br, "meioDeTransporte")
        meio_de_transporte.set_labels(
            "taxi carro-proprio a-pe-triste a-pe-feliz")
        meio_de_transporte.set_prompt_text(
            "Qual é o meio de transporte [taxi, carro-proprio, a-pe-triste, a-pe-
feliz]]?")

        clima = RuleVariable(self.br, "clima")
        clima.set_labels("ruim bom")
        clima.set_prompt_text("Como está o clima [bom, ruim]?")

        c_equals = Condition("=")
        c_more_then = Condition(">")
        c_less_than = Condition("<")

        Regra01 = Rule(self.br, "Regra 01",
                       [Clause(distancia, c_more_then, "5")],
                       Clause(deslocamento, c_equals, "carro"))
```

```
        Regra02 = Rule(self.br, "Regra 02",
                       [Clause(distancia, c_more_then, "1"),
                        Clause(tempo, c_less_than, "15")],
                       Clause(deslocamento, c_equals, "carro"))

        Regra03 = Rule(self.br, "Regra 03",
                       [Clause(distancia, c_more_then, "1"),
                        Clause(tempo, c_more_then, "15")],
                       Clause(deslocamento, c_equals, "a-pe"))

        Regra04 = Rule(self.br, "Regra 04",
                       [Clause(deslocamento, c_equals, "carro"),
                        Clause(local_do_cinema, c_equals, "centro")],
                       Clause(meio_de_transporte, c_equals, "taxi"))

        Regra05 = Rule(self.br, "Regra 05",
                       [Clause(deslocamento, c_equals, "carro"),
                        Clause(local_do_cinema, c_equals, "bairro")],
                       Clause(meio_de_transporte, c_equals, "carro-proprio"))

        Regra06 = Rule(self.br, "Regra 06",
                       [Clause(deslocamento, c_equals, "a-pe"),
                        Clause(clima, c_equals, "ruim")],
                       Clause(meio_de_transporte, c_equals, "a-pe-triste"))

        Regra07 = Rule(self.br, "Regra 07",
                       [Clause(deslocamento, c_equals, "a-pe"),
                        Clause(local_do_cinema, c_equals, "bom")],
                       Clause(meio_de_transporte, c_equals, "a-pe-feliz"))
        return self.br

    def demo_fc(self, LOG):
        LOG.append(
            " --- Ajustando valores para Transporte/Cinema para demo ForwardChain -
--")
        self.br.set_variable_value("distancia", "2")
        self.br.set_variable_value("deslocamento", None)
        self.br.set_variable_value("tempo", "10")
        self.br.set_variable_value("localDoCinema", "centro")
        self.br.set_variable_value("meioDeTransporte", None)
        self.br.set_variable_value("clima", "ruim")
        self.br.display_variables(LOG)

    def demo_bc(self, LOG):
        LOG.append(
```

```
            " --- Ajustando valores para Transporte/Cinema para demo BackwardChain
---")
        self.br.set_variable_value("distancia", None)
        self.br.set_variable_value("deslocamento", None)
        self.br.set_variable_value("tempo", None)
        self.br.set_variable_value("localDoCinema", "centro")
        self.br.set_variable_value("meioDeTransporte", None)
        self.br.set_variable_value("clima", "ruim")
        self.br.display_variables(LOG)
```

Os dois últimos blocos de código, linhas abaixo de “def demo_fc(self, LOG):” e de “def demo_bc(self, LOG):”, apresentam os valores de premissa iniciais. É possível que esses valores sejam alterados e o estudante é encorajado a realizar essas alterações para que verifique o comportamento final do software com os estados iniciais alterados.

O código fonte que implementa o menu de opções, exibido abaixo, dá uma ideia das funcionalidades recodificadas em Python e como elas podem ser chamadas.

```
class APP:
    def __init__(self, app):
        try:
            self.br = None
            self.ref_BR = None
            self.LOG = []
            self.LOG.append(app)
        except Exception as e:
            print("Exception lançada")
            print(e)

    def add_rule_base(self, ref_br):
        self.ref_BR = ref_br

    def show_log(self):
        for x in self.LOG:
            print(x, end=' ')

    def show_log_new_line(self):
        for x in self.LOG:
            print(x, end='\n')

    def clear_text(self):
        self.LOG.clear()
```

```
        self.LOG.append("")
        self.br.reset()

    def menu(self):
        naoCarregada = False
        while True:
            print('\n              : [0] sair', end='\n')
            print('        REGRAS: [1] carregar, [2] mostrar', end='\n')
            print(' FORWARD-CHAIN: [3] carregar variaveis, [4] executar, [7]
reset', end='\n')
            print('BACKWARD-CHAIN: [5] carregar variaveis, [6] executar, [7]
reset', end='\n')
            print('Digite sua opcao: ')
            option = int(input())

            if option == 0:
                return
            elif option == 1:
                if not naoCarregada:
                    self.br = self.ref_BR.create()
                    naoCarregada = True
                else:
                    print("Base de regas já foi carregada!!")
            elif option == 2:
                self.br.display_rules(self.LOG)
                self.show_log()
                self.LOG = []
            elif option == 3:
                self.ref_BR.demo_fc(self.LOG)
                self.show_log_new_line()
                self.LOG = []
            elif option == 4:
                self.LOG.append(" --- Starting Inferencing Cycle --- ")
                self.br.forward_chain()
                self.br.display_variables(self.LOG)
                self.show_log_new_line()
                self.LOG = []
            elif option == 5:
                self.ref_BR.demo_bc(self.LOG)
                self.show_log_new_line()
                self.LOG = []
            elif option == 6:
                goal_list = self.ref_BR.get_goal_list()
                print("Informe a variavel objetivo ", goal_list)
                goal = input()
```

```
            goal_var = self.br.get_variable(goal)
            if goal_var is not None:
                self.br.backward_chain(goal)
            else:
                self.LOG.append("goalVar.__name__: NAO ENCONTEADA")
            self.LOG.append(" --- Ending Inferencing Cycle --- ")
            if goal_var is not None:
                result = goal_var.get_value()
            else:
                result = "DESCONHECIDO"
            self.LOG.append("RESULTADO: " + str(result))
            self.br.display_variables(self.LOG)
            self.show_log_new_line()
            self.LOG = []
        elif option == 7:
            self.clear_text()
```

Os demais códigos estão disponíveis em arquivo zip em anexo.

O estudante é incentivado a analisar o conteúdo dos dois arquivos e buscar entender cada uma das linhas. Se for necessário o estudante pode acessar alguns tutoriais sobre a linguagem de programação Python pelos links abaixo:

https://www.w3schools.com/python/

https://www.python.org/about/gettingstarted/

https://www.codecademy.com/learn/learn-python

Bibliografia

BIGUS, Joseph P.; BIGUS, Jennifer. Constructing intelligent agents using Java. 2001. John Wiley & Sons; 2nd edição.

SCALABRIN, Edson E. Material de apoio para a disciplina “Engenharia de Conhecimento”. 2019. PUCPR.