

IF4070 Representasi Pengetahuan dan Penalaran

Ripple Down Rules Sistem Diagnosa Gangguan Mental



Disusun Oleh:

13521076 Moh. Aghna Maysan Abyan

13521077 Husnia Munzayana

13521115 Shelma Salsabila

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024**


A. Ripple Down Rule

Ripple Down Rule (RDR) adalah sebuah metode untuk melakukan akuisisi pengetahuan, yaitu transfer pengetahuan dari ahli (*expert*) kepada sistem berbasis pengetahuan (Knowledge-Based System/KBS). Pada *framework* RDR, pengetahuan ahli diperoleh berdasarkan konteksnya dan ditambah secara inkremental.

Secara metode, RDR memiliki struktur data dan skenario akuisisi pengetahuan. Pengetahuan ahli disimpan pada struktur data, dengan pengetahuan tersebut dikodekan sebagai sebuah rangkaian aturan. Proses transfer pengetahuan ahli ke KBS dalam RDR terdapat pada skenario akuisisi pengetahuan, yaitu situasi dimana ahli memberikan kasus baru pada sistem dan kemudian menambahkan aturan baru untuk membenarkan klasifikasi apabila ada kesalahan. Ketika ada aturan yang disusun oleh ahli, kondisi aturan ini harus memenuhi kasus-kasus yang salah sebelumnya dan aturan tersebut juga harus tidak memenuhi kasus-kasus yang benar sebelumnya.

B. Permasalahan yang akan diselesaikan

Dalam tugas besar ini, akan dikembangkan sebuah sistem berbasis Ripple Down Rules (RDR) untuk menyelesaikan permasalahan dalam menentukan jenis gangguan mental berdasarkan gejala yang dialami. Proyek ini mencakup 30 jenis gangguan mental yang telah teridentifikasi bersama dengan gejala-gejalanya. Data terkait diperoleh melalui observasi mendalam dari berbagai sumber di internet.

Data yang terkumpul akan diolah dan dimasukkan ke dalam program untuk membentuk struktur RDR yang sistematis dan terorganisir. Hal ini diharapkan dapat membantu proses identifikasi gangguan mental secara lebih efektif, dengan mengandalkan gejala yang diberikan sebagai masukan. Detail lengkap mengenai jenis gangguan mental beserta gejalanya dapat dilihat melalui tautan  Psychological Disorder.xlsx atau drive berikut:

<https://drive.google.com/file/d/1Qh6lybiYPQuWL0IFuD71 Js9fc3xL6QO/view?usp=sharing>

C. Cara Menjalankan Program

Pada kode pemrograman ini, ada tiga library utama yang perlu diinstal. Berikut adalah kegunaan masing-masing library tersebut:

1. **Graphviz:**
 - Berfungsi untuk membantu memvisualisasikan struktur **Ripple Down Rules** dalam bentuk grafik yang mudah dipahami.
2. **Ipywidgets:**
 - Digunakan untuk membuat antarmuka interaktif dalam **Google Colab**.
 - Untuk pengguna **Jupyter Notebook** atau **VSCode**, interaksi dapat dilakukan langsung melalui terminal biasa, sehingga Ipywidgets hanya diperlukan di Colab.
3. **Colorama:**
 - Membantu menampilkan warna pada output terminal untuk membuat teks lebih menarik dan mudah dibaca.

Pastikan ketiga library ini diinstal untuk mendukung fungsionalitas program dengan optimal.

Ada dua cara untuk menjalankan program ini: menggunakan Jupyter Notebook atau Visual Studio Code dan Google Colab. Berikut adalah langkah-langkah yang perlu dilakukan.

1. Menggunakan Jupyter Notebook/Visual Studio Code

Link video tutorial run code di Jupyter Notebook

<https://youtu.be/FoQSLsS5Gxk>

Link video tutorial run code di Visual Studio Code - Setelah Clone

<https://youtu.be/AJqcVDQ99CM>

Adapun tahapan tahapannya adalah sebagai berikut

- Instalasi library
 1. Instal modul Python Graphviz dengan perintah

```
install graphviz
```

Selain itu, instal perangkat lunak Graphviz dari tautan resmi: [Graphviz Download](#).
 2. Instal **Colorama** untuk mempermudah tampilan warna pada output terminal:



```
pip install colorama
```
- Clone repository kode program:

```
git clone  
https://github.com/munzayanahusn/IF4070-Ripple-Down-Rules.git
```

- Kemudian bukan file utamanya pada **folder src** dengan nama **file IF4070_Ripple_Down_Rules.ipynb**
- Jalankan setiap cell secara berurutan atau bisa dengan melakukan run all pada cell

2. Menggunakan Google Colab

Link video tutorial run code di Google Colab
<https://youtu.be/jOW0ZNBB5PE>

- Buka Google Drive berikut:  TUGAS-RPP-RDR
- Buat Shortcut folder tersebut para google drive local Anda.
- Masuk ke tautan berikut:
 IF4070-RDR-13521076-13521077-13521115.ipynb
- Lakukan mounting ke google drive dengan perintah berikut atau run code cell pertama pada Google Colab

```
from google.colab import drive  
drive.mount('/content/drive')
```

- Setelah itu, ubah variabel-variabel yang mengandung file path agar sesuai dengan lokasi file atau shortcut folder di Google Drive. Misalnya:

```
file_path = "/content/drive/MyDrive/TUGAS  
KULIAH/TUGAS-RPP-RDR/mental-illness.rules.txt"  
  
output_dir = "/content/drive/MyDrive/TUGAS  
KULIAH/TUGAS-RPP-RDR"
```

- Kemudian jalankan setiap cell secara berurutan

D. Struktur Rule

Berikut adalah contoh aturan atau pengetahuan yang disimpan di dalam program:

```
True : obj.condition == ['None'] : obj.symptoms == ['None'] : obj.conclusion == Healthy
obj.condition == None : obj.symptoms == None : obj.conclusion == None
obj.condition == ['Memory Loss'] : obj.symptoms == ['Memory Loss'] : obj.conclusion == Narcolepsy
obj.condition == ['Anxiety', 'Social Withdrawal'] : obj.symptoms == ['Anxiety', 'Social Withdrawal'] : obj.conclusion == Specific Phobia
obj.condition == ['Physical Complaints'] : obj.symptoms == ['Anxiety', 'Physical Complaints'] : obj.conclusion == Restless Legs Syndrome
obj.condition == ['physical complaints'] : obj.symptoms == ['social withdrawal', 'physical complaints', 'anxiety'] : obj.conclusion == illness anxiety disorder
obj.condition == ['speech delay'] : obj.symptoms == ['speech delay', 'social withdrawal', 'anxiety'] : obj.conclusion == communication disorder
obj.condition == None : obj.symptoms == None : obj.conclusion == None
obj.condition == None : obj.symptoms == None : obj.conclusion == None
```

Struktur aturan di atas menggunakan format hierarkis untuk memetakan hubungan antara gejala, kondisi, dan diagnosis dalam sistem berbasis aturan. Berikut adalah penjelasan elemen-elemen penting dalam struktur tersebut:

1. Indentasi sebagai Penunjuk Level

Indentasi tambahan menunjukkan level baru atau hierarki aturan. Baris yang lebih ke dalam menggambarkan aturan turunan dari aturan di level sebelumnya. Setiap baris dimulai dengan kondisi tertentu, diikuti oleh gejala yang cocok, dan disimpulkan dengan diagnosis.

2. Baris Setiap Level Indentasi

Pada setiap level dalam hierarki aturan, terdapat dua barisrule. Baris pertama mewakili **left-child**, yang berfungsi sebagai reject-child, yaitu kondisi yang akan dijalankan jika aturan pada level parent tidak terpenuhi. Sementara itu, baris kedua mewakili **right-child**, yang merupakan accept-child yang akan dijalankan jika kondisi pada parent terpenuhi.

3. Komponen Rule

Komponen *rule* yang digunakan pada tugas besar ini memiliki tiga poin utama:

- obj.symptoms:** Daftar gejala yang di-*input* oleh *expoert*, digunakan untuk mencocokkan aturan dan menentukan diagnosis. Setelah data penyakit baru ditambahkan, daftar ini akan mencakup seluruh gejala dari penyakit terkait.
- obj.condition:** Berisi gejala yang lebih spesifik dan tidak tercakup dalam gejala pada aturan parent. Komponen ini digunakan untuk mendalami gejala tertentu yang membutuhkan klarifikasi lebih lanjut untuk menentukan diagnosis.
- obj.conclusion:** Hasil diagnosis yang dihasilkan berdasarkan kecocokan gejala di obj.symptoms dengan aturan dalam sistem.

E. Contoh Input dan Output Program

1) Input Program

- Proses dimulai dengan input yang memungkinkan user (atau expert) untuk memasukkan gejala (symptom) yang relevan dengan penyakit yang sedang dianalisis. Jika terdapat beberapa gejala, *input* setiap gejala dipisahkan oleh koma (,.)

Contoh: Anxiety, Social Withdrawal, Fear of Scrutiny. Lalu, Klik tombol submit.

Enter symptoms below:

Symptoms:

- Kemudian berdasarkan gejala yang dimasukkan, sistem akan mencocokkan data tersebut dengan aturan yang ada dalam sistem Rule-Driven Reasoning (RDR). Jika gejala cocok dengan aturan yang ada, sistem akan menghasilkan sebuah kesimpulan yang menyarankan diagnosis penyakit yang sesuai.
- Setelah kesimpulan dihasilkan, user diminta untuk memvalidasi kesimpulan tersebut dengan memilih antara setuju atau tidak setuju.

Analyzing your input...

Symptoms entered: {'social withdrawal', 'anxiety', 'fear of scrutiny'}
Inferred Conclusion: social anxiety disorder

Do you agree with this conclusion?:

- Jika expert setuju dengan kesimpulan yang diberikan, maka kesimpulan tersebut akan disimpan ke dalam sistem sebagai bagian dari diagnosis yang valid.

Do you agree with this conclusion?:

Thank you for confirming the conclusion!

- Jika expert tidak setuju dengan kesimpulan yang diberikan, sistem akan meminta *expert* untuk memasukkan kesimpulan/penyakit baru terkait gejala yang sebelumnya di-*input*.

Do you agree with this conclusion?:

Yes

No, Enter new illness

Illness:

Submit Illness

- Setelah klik “Submit Illness”, aturan baru (rule) akan ditambahkan. Aturan ini akan dibuat sebagai cabang baru dalam struktur pohon keputusan berdasarkan gejala dan kesimpulan yang diinginkan oleh expert. Aturan baru ini akan disimpan dalam struktur pohon keputusan, di mana setiap simpul (node) berisi kondisi (obj.condition), gejala (obj.symptoms), dan kesimpulan (obj.conclusion) dan disimpan dalam file .rules.
- User atau expert dapat memeriksa gejala lain atau keluar dari program. Jika ahli memilih untuk memeriksa gejala lain, sistem akan menjalankan program kembali.

Do you want to check symptoms again?

Yes, check sympto...

No, exit the program

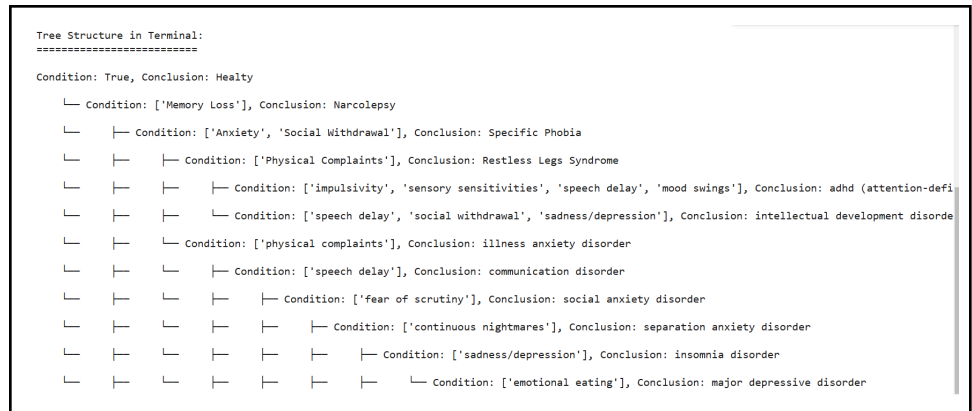
2) Output Program

Output program terdiri dari beberapa hal yakni,

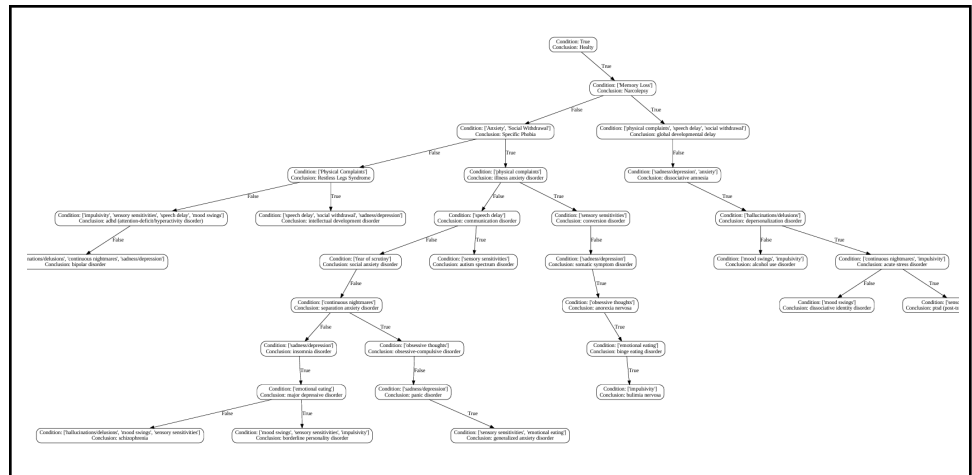
- Hasil inferensi berupa kesimpulan yang dihasilkan berdasarkan gejala (symptom) yang diinputkan oleh user

```
Symptoms entered: {'social withdrawal', 'anxiety', 'fear of scrutiny'}  
Inferred Conclusion: social anxiety disorder
```

- Hasil inferensi tersebut ditampilkan dalam bentuk pohon keputusan yang dapat dilihat langsung dan dicetak di terminal



- Pohon keputusan juga dapat disimpan dalam file PDF untuk mempermudah pemahaman oleh user



- Aturan-aturan yang telah dibuat juga menjadi output serta disimpan dalam file dengan ekstensi .rules

```

True : obj.condition == ['None'] : obj.symptoms == ['None'] : obj.conclusion == Healthy
obj.condition == None : obj.symptoms == None : obj.conclusion == None
obj.condition == ['Memory Loss'] : obj.symptoms == ['Memory Loss'] : obj.conclusion == Narcolepsy
obj.condition == ['Anxiety', 'Social Withdrawal'] : obj.symptoms == ['Anxiety', 'Social Withdrawal'] :
obj.conclusion == Specific Phobia
obj.condition == ['Physical Complaints'] : obj.symptoms == ['Anxiety', 'Physical Complaints'] :
obj.conclusion == Restless Legs Syndrome
obj.condition == ['Impulsivity', 'sensory sensitivities', 'speech delay', 'mood swings'] :
obj.symptoms == ['speech delay', 'anxiety', 'sensory sensitivities', 'mood swings', 'impulsivity'] :
obj.conclusion == adhd (attention-deficit/hyperactivity disorder)
obj.condition == ['speech delay', 'social withdrawal', 'sadness/depression'] : obj.symptoms ==
['speech delay', 'physical complaints', 'social withdrawal', 'sadness/depression'] : obj.conclusion ==
intellectual development disorder
obj.condition == ['physical complaints'] : obj.symptoms == ['social withdrawal', 'physical
complaints', 'anxiety'] : obj.conclusion == illness anxiety disorder
obj.condition == ['speech delay'] : obj.symptoms == ['speech delay', 'social withdrawal',
'anxiety'] : obj.conclusion == communication disorder
obj.condition == ['fear of scrutiny'] : obj.symptoms == ['social withdrawal', 'anxiety', 'fear
of scrutiny'] : obj.conclusion == social anxiety disorder
obj.condition == ['continuous nightmares'] : obj.symptoms == ['social withdrawal',
'anxiety'] : obj.conclusion == separation anxiety disorder
obj.condition == ['sadness/depression'] : obj.symptoms == ['social withdrawal',
'anxiety', 'sadness/depression'] : obj.conclusion == insomnia disorder
obj.condition == None : obj.symptoms == None : obj.conclusion == None

```


F. Penjelasan Kode Program

Kode program yang dibuat terdiri dari beberapa class yaitu,

- 1) Class MentalIllness merepresentasikan gangguan mental melalui dua atribut utama, yaitu symptoms untuk menyimpan daftar gejala dan conclusion sebagai hasil diagnosis berdasarkan gejala yang diberikan. Class ini dilengkapi dengan metode `__str__()` untuk memberikan representasi string objek yang mempermudah tampilan atau debugging, serta metode `reason()` yang menghasilkan penjelasan detail mengenai gangguan mental berdasarkan kondisi atributnya. Adapun kodenya adalah sebagai berikut.

```
class MentalIllness:
    def __init__(self, symptoms=None, conclusion=None):
        self.symptoms = symptoms
        self.conclusion = conclusion

    def __str__(self):
        return 'Mental Illness' + ', '.join([str(att) for att in self.__dict__.values() if
att is not None]) + ']'

    def reason(self):
        return ' and '.join(['{}={}'.format(att, value) for att, value in
self.__dict__.items() if value is not None])
```

- 2) Class Node adalah elemen dalam struktur pohon Ripple Down Rules (RDR) yang berfungsi menyimpan condition sebagai syarat untuk menghasilkan kesimpulan tertentu dalam illness, yang memuat informasi gangguan mental terkait. Selain itu, node ini juga memiliki referensi ke `accept_child` dan `reject_child` untuk membangun cabang pohon berdasarkan hasil evaluasi kondisi. Atribut `parent` digunakan untuk mereferensikan node induk, sementara `depth` mencatat kedalaman node dalam pohon. Metode `__str__()` memberikan representasi deskriptif yang mencakup level, kondisi, detail gangguan mental, dan status hubungan node dengan anak-anaknya. Adapun kodenya adalah sebagai berikut.

```
class Node:
    def __init__(self, parent=None, depth=0, accept_child=None,
reject_child=None, condition=None, illness: MentalIllness = None):
```

```

self.parent = parent
self.depth = depth
self.accept_child = accept_child
self.reject_child = reject_child
self.condition = condition
self.illness = illness

def __str__(self):
    accept_child_str = "Yes" if self.accept_child else "No"
    reject_child_str = "Yes" if self.reject_child else "No"

    illness_details = (
        f'Symptoms: {self.illness.symptoms}, Conclusion:
{self.illness.conclusion}'
        if self.illness else "None"
    )

    return (
        f'Level: {self.depth}, '
        f'Condition: {self.condition}, '
        f'Illness: {illness_details}, '
        f'Has Accept Child: {accept_child_str}, '
        f'Has Reject Child: {reject_child_str}'
    )

```

- 3) Class Tree merepresentasikan struktur pohon Ripple Down Rules (RDR) dengan root sebagai titik awal. Class ini memfasilitasi pengorganisasian node secara hierarkis, di mana setiap node dalam pohon terhubung secara logis melalui referensi parent-child. Metode visualize() memungkinkan penyajian pohon dalam format hierarkis yang deskriptif, menampilkan setiap condition, illness, serta cabang accept_child dan reject_child. Hal ini mendukung pemahaman visual terhadap hubungan antar node dalam struktur RDR. Adapun kodenya adalah sebagai berikut.

```

class Tree:
    def __init__(self, root: Node = None):
        self.root = root

```

```

def visualize(self, node=None):
    if node is None:
        node = self.root

    level = node.depth

    if node.illness:
        print(" " * level + f"Condition: {node.condition}, Symptoms: {node.illness.symptoms}, Conclusion: {node.illness.conclusion}")
    else:
        print(" " * level + f"Condition: {node.condition}, No associated illness")

    if node.reject_child:
        print(" " * (level + 1) + "Reject Child:")
        self.visualize(node.reject_child)
    if node.accept_child:
        print(" " * (level + 1) + "Accept Child:")
        self.visualize(node.accept_child)

```

Selain kelas terdapat beberapa fungsi yang berguna untuk membuat visualisasi dan inferensi. Adapun fungsi-fungsi itu adalah sebagai berikut.

1) Fungsi - fungsi yang digunakan untuk proses visualisasi

| Nama Fungsi | Fungsi |
|-----------------------------------|---|
| def parse_file_rules(file_path) | Berfungsi untuk membaca file aturan (.rules) dan mengonversinya menjadi struktur pohon dengan memisahkan kondisi, kesimpulan, dan hierarki levelnya |
| def print_tree(node, prefix="") | Menampilkan struktur pohon dalam bentuk ASCII di terminal, termasuk kondisi dan kesimpulan setiap node dengan indentasi yang menunjukkan hierarki |
| def build_graph(tree, graph=None, | Membuat visualisasi pohon |

| | |
|--|---|
| parent=None, node_id=0, is_right=True):] | menggunakan Graphviz, menghasilkan grafik pohon yang mencerminkan struktur Ripple Down Rules |
| def mainVisualize(file_path) | Fungsi utama untuk membaca file aturan, mencetak struktur pohon di terminal, dan menghasilkan visualisasi grafis dalam format PDF |

Adapun kodenya adalah sebagai berikut.

```
# RDR Graph Visualization
def parse_file_rules(file_path):
    with open(file_path, "r") as file:
        lines = [line.rstrip() for line in file if line.strip()]
    def extract_condition_conclusion(text):
        parts = text.split(':')
        condition = parts[0].strip()
        conclusion = parts[-1].strip() if len(parts) > 1 else None
        if "obj.condition == " in condition:
            condition = condition.split("obj.condition == ")[1].strip()
        else:
            condition = condition.strip()
        if conclusion and "obj.conclusion == " in conclusion:
            conclusion = conclusion.split("obj.conclusion == ")[1].strip()
        is_none = (condition == "None" or not condition) and (conclusion == "None"
or not conclusion)
        display_text = f"Condition: {condition}, Conclusion: {conclusion}" if not
is_none else "None"
        return {
            'condition': condition,
            'conclusion': conclusion,
            'is_none': is_none,
            'display_text': display_text,
            'original': text.strip()
        }
    first_line = lines[0]
    node_data = extract_condition_conclusion(first_line)
```

```

tree = {
    "data": node_data,
    "children": [],
    "level": 0,
    "position": 0
}
stack = [(tree, len(first_line) - len(first_line.lstrip()))]
current_level = 0
for line in lines[1:]:
    indent = len(line) - len(line.lstrip())
    node_data = extract_condition_conclusion(line)
    if indent > stack[-1][1]:
        current_level += 1
    while stack and indent <= stack[-1][1]:
        stack.pop()
        current_level -= 1
    node = {
        "data": node_data,
        "children": [],
        "level": current_level,
        "position": len(stack[-1][0]["children"])
    }

    if stack:
        parent = stack[-1][0]
        parent["children"].append(node)

    stack.append((node, indent))
return tree

def print_tree(node, prefix=""):
    if not node["data"]["is_none"]:
        print(f"\n{prefix}{node['data']['display_text']}")

    children = node.get("children", [])
    for i, child in enumerate(children):
        if i == len(children) - 1:

```

```

        next_prefix = prefix + "    └── "
    else:
        next_prefix = prefix + "    ┌── "
    print_tree(child, next_prefix)

def build_graph(tree, graph=None, parent=None, node_id=0, is_right=True):
    if graph is None:
        graph = Digraph(comment="Decision Tree", format="pdf")
        graph.attr(rankdir="TB")
        graph.attr('node', shape='box')
        graph.attr(nodesep='1.0')
        graph.attr(ranksep='0.8')

    current_id = f"node{node_id}"

    if not tree["data"]["is_none"]:
        label = f"Condition: {tree['data']['condition']}\nConclusion: {tree['data']['conclusion']}"
        graph.node(current_id, label, style='rounded')
        if parent:
            edge_label = "False" if is_right else "True"
            graph.edge(parent, current_id, label=edge_label)
        else:
            graph.node(current_id, "", style='invis')
            if parent:
                graph.edge(parent, current_id, style='invis')

    children = tree.get("children", [])
    for i, child in enumerate(children):
        child_id = node_id * 10 + i + 1
        # Determine if the child node should be on the right
        next_is_right = (i % 2 == 0)
        build_graph(child, graph, current_id, child_id, next_is_right)

    return graph

def mainVisualize(file_path):

```

```

output_dir = "/content/drive/MyDrive/TUGAS KULIAH/TUGAS-RPP-RDR"
if not os.path.exists(output_dir):
    print(Style.BRIGHT + Fore.RED + f"\nOutput directory does not exist:
{output_dir}" + Style.RESET_ALL)
    ask_to_continue()
    return

if os.path.exists(file_path):
    tree = parse_file_rules(file_path)

    print("\nTree Structure in Terminal:")
    print("=====")
    print_tree(tree)

    print("\nGenerating Visual Graph...")
    graph = build_graph(tree)
    graph.attr(layout='dot')
    graph.attr(ordering='out')

    output_path = os.path.join(output_dir, "final_rules")
    graph.render(output_path, format="pdf", view=True)
    print(f"\nDecision tree saved as PDF at: {output_path}.pdf")
else:
    print(Style.BRIGHT + Fore.RED + "\nFile .rules not found. Please check the
file path." + Style.RESET_ALL)
    ask_to_continue()

```

2) Fungsi - fungsi yang digunakan untuk membaca tree dan membangun tree untuk membantu pembangunan rules dan proses inference

| Nama Fungsi | Fungsi |
|------------------------|---|
| def clean_value(value) | Berfungsi untuk membersihkan nilai input dari tanda baca tambahan seperti spasi ekstra, sehingga data menjadi lebih rapi dan dapat digunakan. |

| | |
|----------------------------|--|
| def parse_rules(file_path) | Berfungsi untuk membaca file aturan (.rules), memproses baris-barisnya menjadi struktur pohon Ripple Down Rules (RDR), dan membangun hierarki node dengan kondisi, gejala, serta kesimpulan. |
|----------------------------|--|

Adapun kodenya adalah sebagai berikut.

```
# Read Rules and Build Tree
def clean_value(value):
    value = value.strip()
    if value.startswith '[' and value.endswith(']'):
        value = value[1:-1].strip()
    value = value.replace('""', '').replace("'", '').strip()
    return value

def parse_rules(file_path):
    with open(file_path, 'r') as file:
        lines = file.readlines()

    root = None
    node_stack = []
    is_right_child = []

    for i, line in enumerate(lines):
        line = line.replace("  ", "\t")

        if not line.strip():
            continue

        level = line.count('\t')
        line = line.lstrip('\t')

        if len(is_right_child) <= level:
            is_right_child.extend([False] * (level + 1 - len(is_right_child)))

        if level == 0:
```



```

line = line.replace("True : ", "")

# Parse the line content
parts = line.split(" : ")
if len(parts) < 3:
    print(f"Skipping invalid line: {line}")
    if i == len(lines) - 1:
        break
    continue

# Extract condition, symptoms, and conclusion
condition_part = parts[0].replace("obj.condition == ", "").strip()
symptoms_part = parts[1].replace("obj.symptoms == ", "").strip()
conclusion_part = parts[2].replace("obj.conclusion == ", "").strip()

# Clean the values
condition = clean_value(condition_part)
symptoms = clean_value(symptoms_part)
conclusion = clean_value(conclusion_part)

# Handle None conclusion cases before printing
if conclusion == "None":
    is_right_child[level] = not is_right_child[level]
    if i == len(lines) - 1:
        break
    continue

# Create a MentalIllness object and Node
illness = MentalIllness(
    symptoms=symptoms.split(", ") if symptoms else None,
    conclusion=conclusion
)

current_node = Node(
    condition=condition.split(", ") if condition else None,
    illness=illness,
    depth=level

```

```

)

# Build parent-child relationships
if level == 0:
    root = current_node
    node_stack = [current_node]
else:
    while len(node_stack) > level:
        node_stack.pop()

    parent_node = node_stack[-1]

    if not is_right_child[level] and parent_node.reject_child is None:
        parent_node.reject_child = current_node
    elif is_right_child[level] and parent_node.accept_child is None:
        parent_node.accept_child = current_node

    node_stack.append(current_node)

    is_right_child[level] = not is_right_child[level]

return Tree(root=root)

```

3) Fungsi - fungsi yang digunakan untuk menghasilkan inference dari knowledge yang diketahui

| Nama Fungsi | Fungsi |
|---|---|
| def get_inference(tree, input_symptoms) | Berfungsi untuk menelusuri pohon Ripple Down Rules berdasarkan gejala yang dimasukkan oleh pengguna, mengevaluasi kondisi setiap node, dan mengembalikan kesimpulan akhir (<i>last true</i>) yang sesuai beserta node aktif terakhir. |
| def on_button_click_with_inference(b) | Berfungsi untuk menangani input gejala dari pengguna saat tombol |

| | |
|--|---|
| | "Submit" ditekan, memanggil fungsi inferensi untuk menentukan kesimpulan, dan menampilkan hasilnya dengan opsi untuk menerima atau menolak kesimpulan tersebut. |
|--|---|

Adapun kodenya adalah sebagai berikut.

```
# Get Inference/Conclusion based on Current Rules/Knowledges
def get_inference(tree, input_symptoms):
    current_node = tree.root
    last_true = tree.root
    last_active = tree.root

    while current_node:
        # print("\nNext")
        # print("current_node", current_node)
        # print("last_true", last_true)
        # print("last_active", last_active)

        if current_node.condition:
            if current_node == tree.root:
                condition_met = True
            else:
                condition_met = set(cond.lower() for cond in
current_node.condition).issubset(input_symptoms)

        # print("Condition met", condition_met)
        if condition_met:
            last_true = current_node
            last_active = current_node
            if current_node.accept_child:
                current_node = current_node.accept_child
            else:
                return last_true.illness.conclusion, last_active
        else:
            last_active = current_node
```

```

        if current_node.reject_child:
            current_node = current_node.reject_child
        else:
            return last_true.illness.conclusion, last_active
    else:
        if last_true.illness and last_true.illness.conclusion:
            return last_true.illness.conclusion, last_active
        break

    return "No matching conclusion found for the provided symptoms."

def ask_to_continue():
    def on_yes_click(b):
        # clear_output(wait=True)
        print(Style.BRIGHT + Fore.YELLOW + "Restarting symptom check..." +
              Style.RESET_ALL)
        main()

    def on_no_click(b):
        print(Style.BRIGHT + Fore.GREEN + "Exiting the program. Thank you!" +
              Style.RESET_ALL)
        global continue_checking
        continue_checking = False

    # Create "Yes" and "No" buttons
    yes_button = widgets.Button(description="Yes, check symptoms again")
    no_button = widgets.Button(description="No, exit the program")

    yes_button.on_click(on_yes_click)
    no_button.on_click(on_no_click)

    print(Style.BRIGHT + Fore.YELLOW + "\nDo you want to check symptoms
again?" + Style.RESET_ALL)
    button_box = HBox([yes_button, no_button])
    display(button_box)

def on_button_click_with_inference(b):

```

```

global last_active
global user_symptoms
global continue_checking

print(Style.BRIGHT + Fore.YELLOW + "\nAnalyzing your input..." +
Style.RESET_ALL)
user_symptoms = set(symptom.strip().lower() for symptom in
symptoms_input.value.split(",") if symptom.strip())
print(f"\nSymptoms entered: {user_symptoms}")

if not mental_health_tree:
    print(Style.BRIGHT + Fore.RED + "\nMental health tree not initialized" +
Style.RESET_ALL)
    ask_to_continue()

elif not user_symptoms:
    print(Style.BRIGHT + Fore.RED + "\nNo symptoms entered. Please input
symptoms" + Style.RESET_ALL)
    ask_to_continue()

else:
    conclusion, last_active = get_inference(mental_health_tree,
user_symptoms)
    print(Style.BRIGHT + Fore.GREEN + f"\nInferred Conclusion: {conclusion}" +
Style.RESET_ALL)

    # Display the question
    if conclusion is not None :
        print(Style.BRIGHT + Fore.YELLOW + "\nDo you agree with this
conclusion?:" + Style.RESET_ALL)

        accept_button = widgets.Button(description="Yes")
        reject_button = widgets.Button(description="No, Enter new illness")
        accept_button.on_click(accept_inference)
        reject_button.on_click(reject_inference)

        button_box = HBox([accept_button, reject_button])

```

```
display(button_box)
```

4) Fungsi yang digunakan untuk meng-expand tree untuk menambah kesimpulan *illness* yang baru

| Nama Fungsi | Fungsi |
|--|--|
| <pre>def expand_tree(tree, input_symptoms, input_illness, last_active)</pre> | Berfungsi untuk memperluas pohon Ripple Down Rules dengan menambahkan aturan baru berdasarkan gejala yang dimasukkan pengguna dan kondisi terakhir yang aktif, kemudian menyimpan dan memvisualisasikan pembaruan pohon. |

Adapun kodenya adalah sebagai berikut.

```
# Expand Tree to Add Illness/Rule
def expand_tree(tree, input_symptoms, input_illness, last_active) :
    # print("\nEXPAND TREE")
    # print("last_active", last_active)
    # print("input_symptoms", input_symptoms)

    last_active_symptoms = set(symptom.strip().lower() for symptom in
last_active.illness.symptoms if symptom.strip())
    if last_active.illness and last_active.illness.symptoms:
        condition_diff = set(input_symptoms) - last_active_symptoms
    else:
        condition_diff = set(input_symptoms)

    if last_active == tree.root:
        condition_met = True
    else:
        condition_met = set(cond.lower() for cond in
last_active.condition).issubset(input_symptoms)

    # Create new node
```

```

new_illness = MentalIllness(
    symptoms = list(input_symptoms) if input_symptoms else None,
    conclusion = input_illness if isinstance(input_illness, str) else
next(iter(input_illness), None)
)

new_node = Node(
    condition=list(condition_diff) if condition_diff else None,
    illness=new_illness,
    depth=last_active.depth + 1
)

# Assign new node
if condition_met :
    if last_active.accept_child :
        print("Condition met but Accept Child already exists")
    else :
        last_active.accept_child = new_node
else :
    if last_active.reject_child :
        print("Condition not met but Reject Child already exists")
    else :
        last_active.reject_child = new_node

open(file_path, "w").close()

# Save the tree
save_tree(mental_health_tree.root, file_path)
print(f"\nTree saved to {file_path}")

mainVisualize(file_path)

ask_to_continue()

```

- 5) Fungsi yang digunakan untuk menyimpan rule-rule yang telah terbentuk ke ekstensi .rule

| Nama Fungsi | Fungsi |
|---|---|
| def save_tree(node, file_path, level=0, is_first_call=True) | Fungsi save_tree menyimpan representasi rekursif dari pohon keputusan ke dalam file, mencatat kondisi, gejala, dan kesimpulan setiap simpul, serta mendukung penambahan simpul anak jika ada. |

Adapun kodenya adalah sebagai berikut.

```
# Save new Tree
def save_tree(node, file_path, level=0, is_first_call=True):
    mode = "w" if is_first_call else "a"

    with open(file_path, mode) as file:
        if is_first_call:
            file.write("True : ")

        indent = "  " * level

        # Format node properties
        if node is None or (node.condition is None and node.illness is None):
            file.write(f"{indent}obj.condition == None : obj.symptoms == None :
obj.conclusion == None\n")
        else:
            condition = f"obj.condition == {node.condition}" if node.condition else
"obj.condition == None"
            if node.illness:
                symptoms = f"obj.symptoms == {node.illness.symptoms}" if
node.illness.symptoms else "obj.symptoms == None"
                conclusion = f"obj.conclusion == {node.illness.conclusion}" if
node.illness.conclusion else "obj.conclusion == None"

            else:
                symptoms = "obj.symptoms == None"
                conclusion = "obj.conclusion == None"
```



```

# Write the formatted line
file.write(f"{indent}{condition} : {symptoms} : {conclusion}\n")

# Recursively process children if they exist
if node and (node.accept_child or node.reject_child):
    # Process reject_child
    if node.reject_child:
        save_tree(node.reject_child, file_path, level + 1, False)
    else:
        with open(file_path, "a") as f:
            f.write(f"{indent}  obj.condition == None : obj.symptoms == None :
obj.conclusion == None\n")

    # Process accept_child
    if node.accept_child:
        save_tree(node.accept_child, file_path, level + 1, False)
    else:
        with open(file_path, "a") as f:
            f.write(f"{indent}  obj.condition == None : obj.symptoms == None :
obj.conclusion == None\n")

```

6) Fungsi yang digunakan untuk menerima/menolak adanya *ilness* baru

| Nama Fungsi | Fungsi |
|-------------------------|---|
| def accept_inference(b) | Fungsi ini dijalankan ketika ahli menerima kesimpulan yang diajukan, menampilkan pesan terima kasih yang menunjukkan bahwa kesimpulan telah diterima dengan baik, kemudian memanggil fungsi ask_to_continue() untuk melanjutkan proses. |
| def reject_inference(b) | Fungsi ini dijalankan ketika ahli menolak kesimpulan yang diajukan. Sebuah input teks muncul untuk memungkinkan ahli memasukkan nama penyakit yang mungkin sesuai |

| | |
|-------------------------|---|
| | dengan gejala yang terdeteksi. Setelah penyakit dimasukkan, fungsi |
| def submit_illness(btn) | submit_illness(btn) dipanggil untuk memproses input tersebut. Jika penyakit yang valid dimasukkan, sistem akan memperbarui pohon keputusan dengan informasi penyakit baru menggunakan fungsi expand_tree(). Jika tidak ada penyakit yang dimasukkan, pesan kesalahan ditampilkan dan meminta pengguna untuk mencoba lagi. |

Adapun kodenya adalah sebagai berikut.

```
# Expert Accept/Reject Conclusion
def accept_inference(b):
    print(Style.BRIGHT + Fore.GREEN + "\nThank you for confirming the
conclusion!" + Style.RESET_ALL)
    ask_to_continue()

def reject_inference(b):
    print("\n")
    illness_input = widgets.Text(
        value="",
        placeholder='Enter illness name here...',
        description='Illness:',
        disabled=False,
        layout=widgets.Layout(width='600px')
    )

    submit_illness_button = widgets.Button(description="Submit Illness")

    def submit_illness(btn):
        new_illness = set(illness.strip().lower() for illness in
illness_input.value.split(",") if illness.strip())
        if new_illness:
            print(f"\nNew illness entered: {new_illness}")
            expand_tree(mental_health_tree, user_symptoms, new_illness,
```

```
last_active)
    else:
        print(Style.BRIGHT + Fore.RED + "\nNo illness entered. Please try
again." + Style.RESET_ALL)

        ask_to_continue()

submit_illness_button.on_click(submit_illness)
display(illness_input, submit_illness_button)
```