

Laporan Penjelasan Implementasi *Ripple Down Rule*

IF4070 Representasi Pengetahuan dan Penalaran



Disusun Oleh:

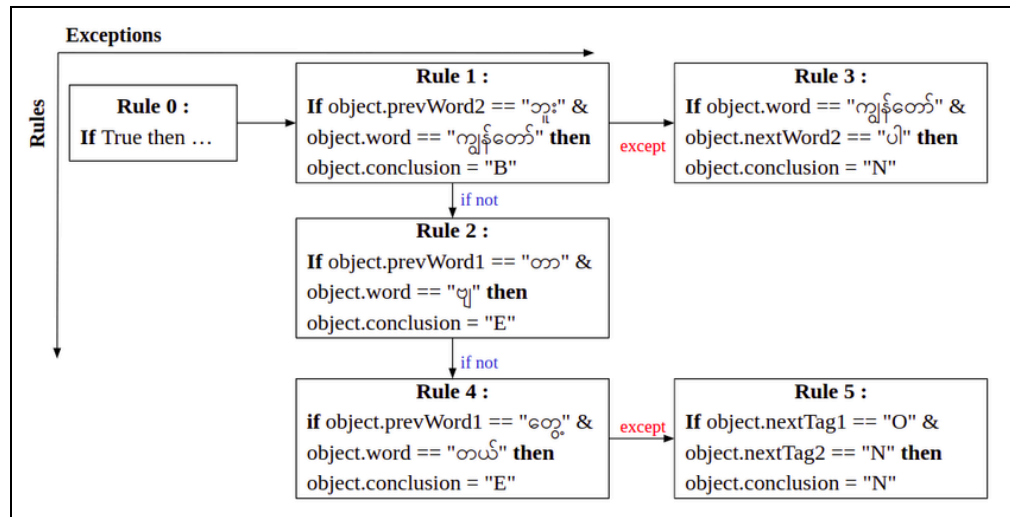
Jason Rivalino	13521008
Melvin Kent Jonathan	13521052
Juan Christopher Santoso	13521116

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

2024

I. Pendahuluan

Ripple Down Rules adalah sebuah sistem kepakaran yang digunakan untuk membangun pengetahuan secara bertahap. *Ripple Down Rules* dirancang sebagai sebuah sistem yang *robust* terhadap perubahan dengan berbasis penambahan pengetahuan secara *incremental*.



Gambar 1.1 Visualisasi *Binary Ripple Down Rules*

Sumber: [mySentence: Sentence Segmentation for Myanmar Language using Neural Machine Translation Approach](#)

Ripple Down Rules umumnya menggunakan struktur *tree* dalam membangun basis pengetahuannya. Setiap pengetahuan diwakili oleh *node* yang terdapat pada *tree* tersebut. Berdasarkan pencabangannya, *Ripple Down Rules* dibagi menjadi dua jenis, yakni *Binary Ripple Down Rules* dan *Multiple Classification Ripple Down Rules*. Sesuai namanya, *Binary Ripple Down Rules* memilih maksimal dua pencabangan, yakni cabang *True* dan cabang *False*. Cabang-cabang ini adalah jalur yang akan dilakukan pengecekan ketika sistem digunakan, bergantung kepada apakah *rule* yang dikandung pada node tertentu bernilai *True* atau *False*. Di sisi lain, *Multiple Classification Ripple Down Rule* memiliki berbagai percabangan bergantung pada banyaknya percabangan klasifikasi yang dibutuhkan.

Pada tugas ini, sistem *Ripple Down Rules* yang diimplementasikan dalam tugas ini adalah *Binary Ripple Down Rules*. Tak hanya itu, domain yang akan kami gunakan dalam pengerjaan tugas ini adalah domain pengajuan asuransi. Maka dari itu, kami

mengimplementasikan sebuah sistem *Binary Ripple Down Rules* untuk melakukan klasifikasi terkait proses pengajuan asuransi yang diperlukan berdasarkan karakteristik/ciri-cirinya.

II. Implementasi

1. Struktur Data

Struktur data yang digunakan dalam implementasi program *Ripple Down Rules* disimpan dalam bentuk kelas (*class*). Terdapat tiga *classes* yang dibentuk dalam pengembangan program ini, yakni: *query*, *node*, dan *RDR*. Fungsionalitas dan atribut dari masing-masing *class* akan dijelaskan sebagai berikut:

a. *Query*

Query adalah *class* yang berguna untuk melakukan *parsing* dan menyimpan input yang dimasukkan oleh pengguna. Terdapat 2 jenis *query* yang dapat dimasukkan pengguna ke dalam sistem RDR, yakni:

1. *Ask query* yang merupakan sebuah pertanyaan. *Query* ini dimasukkan oleh pengguna ketika pengguna ingin mendapatkan sebuah hasil dari *knowledge* yang sudah dibangun oleh sistem RDR.
2. *Add query* yang merupakan sebuah pernyataan. *Query* ini dimasukkan oleh pengguna ketika pengguna ingin menambahkan sebuah *knowledge* atau *rule* ke dalam sistem RDR.

Untuk kelas *query*, atribut yang dikandung adalah sebagai berikut:

Tabel 2.1 Atribut kelas *query*

No	Nama Atribut	Tipe Data	Penjelasan
1	<i>type</i>	<i>string</i>	Atribut <i>type</i> menjelaskan jenis dari <i>query</i> . Nilai yang dikandung dari atribut ini adalah enumerasi dari <i>ask</i> dan <i>add</i> .
2	<i>conditions</i>	<i>list of string</i>	Atribut ini berisikan segala <i>conditions</i> yang diinput oleh pengguna.
3	<i>result</i>	<i>string</i> None	Atribut ini berisikan <i>result</i> dari <i>query</i> bertipe <i>add</i> . Atribut ini bernilai None apabila <i>query</i> bertipe <i>ask</i> .

Format atau notasi yang digunakan untuk membangkitkan sebuah *instance query* adalah sebagai berikut:

- i. *Ask query* : [CONDITIONS] ?
Contoh *input* : A, B, C ?
- ii. *Add query* : [CONDITIONS] > [RESULT]
Contoh *input* : A, B, C > D

b. *Node*

Node adalah *class* yang menyimpan *rule* dalam sistem *Ripple Down Rules*. Mengingat *Ripple Down Rules* memiliki struktur *tree* (akan dijelaskan lebih lanjut pada bagian *class RDR*), *class node* berfungsi sebagai komponen penyusun dari *class RDR* tersebut.

Untuk kelas *query*, atribut yang dikandung adalah sebagai berikut:

Tabel 2.2 Atribut kelas *node*

No	Nama Atribut	Tipe Data	Penjelasan
1	<i>rule_conditions</i>	<i>list of string</i>	Atribut ini menyimpan daftar kondisi dalam pengecekan <i>rule</i> .
2	<i>rule_result</i>	<i>string</i>	Atribut ini menyimpan hasil <i>knowledge</i> yang dikandung apabila pengecekan <i>rule</i> bernilai TRUE.
3	<i>true_node</i>	<i>node</i> None	Atribut ini mengandung <i>node</i> berikutnya yang perlu dikunjungi apabila pengecekan <i>rule</i> bernilai TRUE. Atribut ini bernilai None apabila tidak ada <i>node</i> lanjutan yang perlu dikunjungi.
4	<i>false_node</i>	<i>node</i> None	Atribut ini mengandung <i>node</i> berikutnya yang perlu dikunjungi apabila pengecekan <i>rule</i> bernilai FALSE. Atribut ini bernilai None apabila tidak ada <i>node</i> lanjutan yang perlu dikunjungi.

c. *RDR*

RDR adalah *class* yang merepresentasikan sistem *Ripple Down Rules*. *Ripple Down Rules* memiliki struktur berupa *tree*. Maka dari itu, *class* ini hanya memiliki satu atribut yakni *tree*.

Tabel 2.3 Atribut kelas *RDR*

No	Nama Atribut	Tipe Data	Penjelasan
1	<i>tree</i>	<i>node</i>	Atribut ini merupakan <i>node</i> yang menjadi <i>root</i> bagi struktur <i>tree</i> dalam <i>RDR</i> .

2. Penjelasan Algoritma

Skenario yang umumnya digunakan dalam proses penggunaan sistem *Ripple Down Rules* yang telah dikembangkan dapat dibagi menjadi 6, yakni melakukan *parsing* pada *input* pengguna, melakukan pencarian *result* berdasarkan *input*, menambahkan *rule* baru berdasarkan *input*, menampilkan pohon *RDR*, melakukan *save* terhadap pohon *RDR*, dan melakukan *load* terhadap pohon *RDR* yang disimpan. Berikut adalah penjelasan dari langkah-langkah yang dilakukan pada masing-masing skenario:

a. Pelaksanaan *parsing input* pengguna

1. Penentuan karakter dan pola *input*

Hal pertama yang dilakukan pada bagian *parsing* adalah mengenali jenis *query* yang dimasukkan oleh pengguna. Mengingat terdapat dua *query* yang valid, maka sistem *parsing* perlu dapat mengenali jenis *query* berdasarkan karakter dan pola yang ada. Pada bagian ini, digunakan *regex* untuk membantu mengenali pola yang ada. *Regex* yang digunakan antara lain:

a. *Ask query* dengan pola $r'^{([\w\s\-\.\$\\()]+)(,s?[\w\s\-\.\$\\()]+)^*}\\?$', dan$

b. *Add query* dengan pola

$r'^{([\w\s\-\.\$\\()]*)(,s?[\w\s\-\.\$\\()]+)^*}\\s?>\\s?[\w\s\-\.\$\\()]+$',$

2. Parsing input untuk kondisi pengecekan *rule* (*ask*)

Pada proses *parsing query ask*, hal pertama yang dipisahkan adalah tanda tanya itu sendiri. Tanda tanya hanya digunakan sebagai *identifier* dari *ask query* itu sendiri. Namun, tanda tanya ini tidak memiliki makna apapun. Setelah itu, *string* yang tersisa merupakan *string* untuk data *conditions* yang dipisahkan dengan tanda koma. Maka dari itu, *string* ini dilakukan *split* dengan koma sebagai *separator* dan disimpan dalam sebuah *list*. Nilai *result* diset dengan *None value*.

3. Parsing input untuk kondisi penambahan *rule* (*add*)

Pada proses *parsing query ask*, hal pertama yang dipisahkan adalah panah (>). *Substring* setelah tanda panah akan disimpan sebagai *result* dalam tipe *string*. Di sisi

lain, *substring* sebelum tanda panah akan dilakukan *split* dengan separator koma lalu disimpan dalam sebuah *list* sebagai *conditions*.

b. Pencarian *result* berdasarkan *input*

1. Pemrosesan *input* menjadi *query*

Pada tahap ini, *input* pengguna diproses menggunakan pola *regular expression* yang telah didefinisikan. Apabila memenuhi, objek *query* akan dibentuk dengan tipe “ask” dan *conditions* yang didefinisikan oleh pengguna. Atribut *result* akan dibiarkan menjadi None.

2. Pengecekan pemenuhan *condition* dari *root node* RDR

Instansiasi dari kelas RDR akan melakukan pengecekan terhadap *root node* dari pohon yang dimilikinya. Apabila semua *condition* yang didefinisikan pada *node* tersebut dipenuhi oleh *conditions* yang menjadi masukan dari *method* *check_rule()*, *method* akan memberikan return True. Jika tidak, maka *method* akan memberikan return False.

3. Pembaruan *result*

Apabila pengecekan *rule* menghasilkan True, *result* akan diperbarui menjadi *rule_result* yang telah didefinisikan pada properti *node* tersebut. Apabila pengecekan *rule* menghasilkan False, *result* tidak akan diperbarui.

4. Pengeliminasian *satisfied condition*

Apabila pengecekan *rule* menghasilkan True, kondisi yang terdefinisi pada *rule* dari *node* akan dieliminasi dari *conditions* yang menjadi masukan sehingga menghasilkan *remaining conditions*.

5. Pengecekan *true node* atau *false node*

Apabila pengecekan *rule* menghasilkan True, *remaining conditions* akan diteruskan menjadi masukan dari *true node*. Apabila pengecekan *rule* menghasilkan False, *conditions* (tanpa dieliminasi) akan diteruskan menjadi masukan dari *false node*. Proses pengecekan *true node* atau *false node* ini sama seperti yang telah dijabarkan pada poin-poin sebelumnya dan dilakukan secara rekursif.

6. Pengembalian *result*

Apabila proses rekursif penelusuran pohon telah sampai pada *node* yang tidak memiliki *child node* yang diinginkan (*true node* pada kasus True, *false node* pada kasus False), *result* terbaru akan dikirim menjadi hasil final dari proses pencarian *result*.

c. Penambahan *rule* baru berdasarkan *input*

1. Pemrosesan *input* menjadi *query*

Pada tahap ini, *input* pengguna diproses menggunakan pola *regular expression* yang telah didefinisikan. Apabila memenuhi, objek *query* akan dibentuk dengan tipe “add” dan *conditions* beserta *result* yang didefinisikan oleh pengguna.

2. Pengecekan pemenuhan *condition* dari *root node* RDR

Instansiasi dari kelas RDR akan melakukan pengecekan terhadap *root node* dari pohon yang dimilikinya. Apabila semua *condition* yang didefinisikan pada *node* tersebut dipenuhi oleh *conditions* yang menjadi masukan dari *method* `check_rule()`, *method* akan memberikan return True. Jika tidak, maka *method* akan memberikan return False.

3. Pengeliminasian *satisfied condition*

Apabila pengecekan *rule* menghasilkan True, kondisi yang terdefinisi pada *rule* dari *node* akan dieliminasi dari *conditions* yang menjadi masukan sehingga menghasilkan *remaining conditions*.

4. Pengecekan *true node* atau *false node*

Apabila pengecekan *rule* menghasilkan True, *remaining conditions* beserta *result* akan diteruskan menjadi masukan dari *true node*. Apabila pengecekan *rule* menghasilkan False, *conditions* (tanpa dieliminasi) beserta *result* akan diteruskan menjadi masukan dari *false node*. Proses pengecekan *true node* atau *false node* ini sama seperti yang telah dijabarkan pada poin-poin sebelumnya dan dilakukan secara rekursif.

5. Penambahan *node* baru

Apabila proses rekursif penelusuran pohon telah sampai pada *node* yang tidak memiliki *child node* yang diinginkan (*true node* pada kasus True, *false node* pada kasus False), akan ditambahkan *node* baru pada cabang yang sesuai dengan *remaining conditions* menjadi atribut `rule_conditions`-nya dan *result* menjadi atribut `rule-result`-nya..

d. Penampilan pohon RDR

Mekanisme *print* pohon RDR dilakukan dengan fungsi rekursif. Berikut merupakan penampilan pohon RDR pada terminal.

```
Printing the RDR tree:
TRUE -> b
|- true node: TRUE -> c
|   |- true node: ['a'] -> c
|       |- true node: ['z'] -> ax
|           |- true node: ['huhu'] -> awikwok
|           |- false node: ['c'] -> d
|               |- true node: ['b'] -> d
|               |- false node: ['d'] -> a
|                   |- false node: TRUE -> a
|                       |- true node: ['b'] -> e
|- false node: ['g'] -> c
    |- true node: ['h', 'i'] -> c
    |- false node: ['bn'] -> d
        |- true node: ['agh', 'rt', 'p'] -> d
        |- false node: ['d'] -> a
            |- false node: TRUE -> a
                |- true node: ['b'] -> e
```

e. Penyimpanan struktur pohon RDR

1. Penamaan file untuk struktur pohon RDR yang akan disimpan

Proses penyimpanan struktur pohon RDR ke dalam bentuk file diawali dengan penamaan dokumen file yang akan disimpan. Pengguna dapat memilih nama file untuk penyimpanan struktur pohon RDR ke dalam dua bentuk format file yaitu TXT.

2. Penyimpanan struktur pohon RDR dalam bentuk format file TXT

Proses penyimpanan dalam bentuk TXT dilakukan dengan memanfaatkan proses rekursif dengan kondisi basisnya ketika pohon masih kosong. Jika pohon berisi maka akan menjalankan proses rekursif dengan memanggil fungsi penyimpanan secara berulang-ulang (*save_subtree*) selama kondisi *node* masih memiliki *prefix* dan belum sampai pada ujung *leaf*. Terdapat penanganan ketika input *rule* kosong maka *rule* dianggap benar (TRUE). Hasil pemanggilan ini kemudian disimpan dalam bentuk file TXT.

f. Pemuatan struktur pohon RDR yang disimpan

1. Pemilihan jenis data file untuk menampilkan pohon yang disimpan

Pemilihan jenis data file dilakukan di awal proses ketika pengguna menjalankan program dan memilih untuk melakukan proses *load* untuk pemuatan struktur pohon RDR dengan menggunakan file. Terdapat dua opsi pilihan untuk proses *load* file antara dengan format file TXT. Setelah memilih opsi file, pengguna akan memuat nama file yang akan diproses untuk pemuatan struktur pohon RDR.

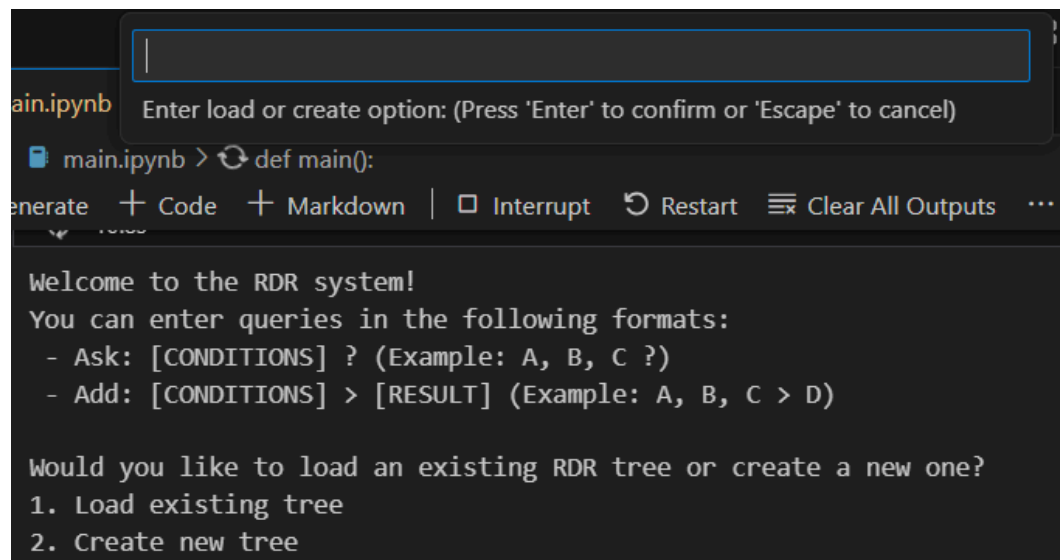
2. Pemuatan struktur pohon RDR melalui format file TXT

Proses pemuatan untuk format file TXT dilakukan dengan proses rekursif melalui basisnya yaitu proses *load* pada susunan pohon di baris pertama. Rekursif dilakukan dengan pengecekan kondisi *node* dan juga penentuan kondisi *conditions* dan *result* dari data pohon RDR dalam file TXT serta penanganan kondisi TRUE. Proses pemanggilan seluruh *node* kemudian dilakukan secara rekursif melalui pemanggilan fungsi berulang hingga mencapai kondisi ketika tidak ditemukan lagi *node* apapun (sudah mencapai dasar *leaf* pohon) dan kemudian *node* dikembalikan untuk diproses.

3. Alur Keseluruhan Program

Berikut merupakan alur secara keseluruhan untuk menjalankan program RDR:

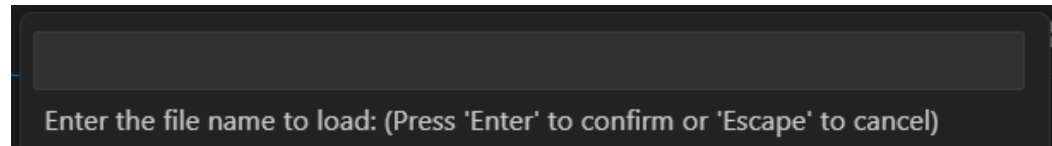
- a. Informasi awal pada program sistem RDR. Pada bagian ini, terdapat opsi pemilihan untuk membuat struktur pohon RDR dari awal ataupun memuat pohon berdasarkan file.



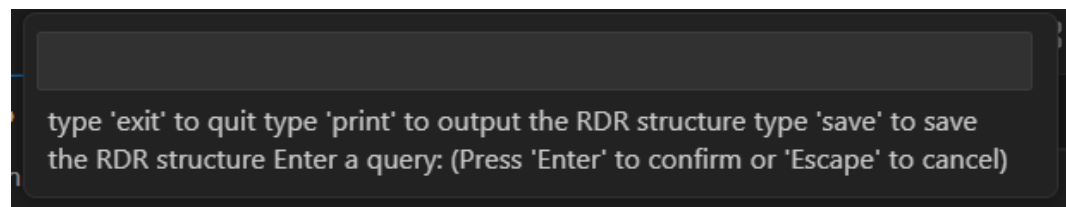
```
main.ipynb > Enter load or create option: (Press 'Enter' to confirm or 'Escape' to cancel)
main.ipynb > def main():
Welcome to the RDR system!
You can enter queries in the following formats:
- Ask: [CONDITIONS] ? (Example: A, B, C ?)
- Add: [CONDITIONS] > [RESULT] (Example: A, B, C > D)

Would you like to load an existing RDR tree or create a new one?
1. Load existing tree
2. Create new tree
```

- b. Jika memilih *load*, pengguna dapat memilih opsi memuat data berdasarkan jenis file antara TXT. Setelahnya pengguna dapat menuliskan nama file yang ingin diproses

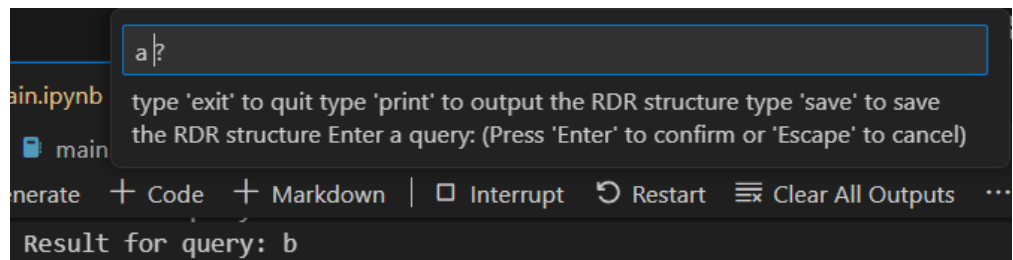


- c. Jika memilih *create*, pengguna dapat langsung melakukan penambahan aturan baru melalui input

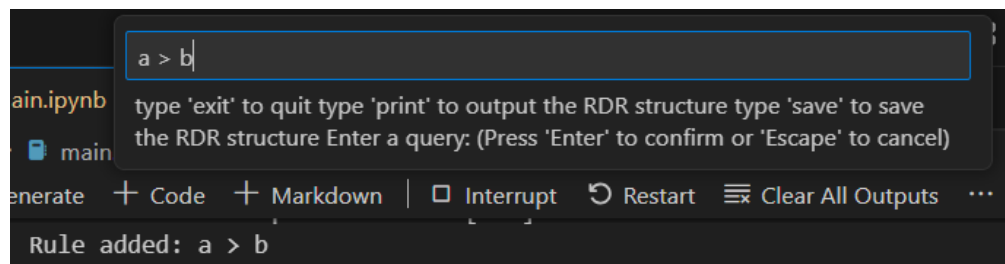


- d. Terdapat beberapa *command* yang dapat digunakan pada menu utama ini, antara lain sebagai berikut:

1. Pengecekan kondisi aturan dengan format penulisan [CONDITIONS] ?



2. Penambahan aturan baru dengan format penulisan [CONDITIONS] > [RESULT]



3. print: bertujuan untuk menampilkan *output* dari struktur pohon RDR yang sudah dibuat

```
print
type 'exit' to quit type 'print' to output the RDR structure
the RDR structure Enter a query: (Press 'Enter' to confirm
generate + Code + Markdown | Interrupt Restart
Printing the RDR tree:
TRUE -> b
|- true node: TRUE -> c
|   |- true node: ['a'] -> c
|   |   |- true node: ['z'] -> ax
|   |   |   |- true node: ['huhu'] -> awikwok
|   |   |   |- false node: ['c'] -> d
|   |   |       |- true node: ['b'] -> d
|   |   |       |- false node: ['d'] -> a
|   |   |           |- false node: TRUE -> a
|   |   |               |- true node: ['b'] -> e
|   |- false node: ['g'] -> c
|   |- true node: ['h', 'i'] -> c
|   |- false node: ['bn'] -> d
|       |- true node: ['agh', 'rt', 'p'] -> d
|       |- false node: ['d'] -> a
|       |- false node: TRUE -> a
|           |- true node: ['b'] -> e
```

4. save: berfungsi untuk penyimpanan struktur *tree* yang sudah di buat ke dalam bentuk format file TXT. Pengguna dapat memasukkan nama file untuk penyimpanan

```
save
type 'exit' to quit type 'print' to output the RDR structure type 'save' to save
the RDR structure Enter a query: (Press 'Enter' to confirm or 'Escape' to cancel)
```

```
pohon_rdr
Enter the file name to save: (Press 'Enter' to confirm or 'Escape' to cancel)
```

```
test\txt
  insurance_claim_knowledge.txt
  pohon_rdr.txt
```

5. exit: berfungsi untuk keluar dari program

```
exit
type 'exit' to quit type 'print' to output the RDR structure type 'save' to save the RDR structure Enter
a query: (Press 'Enter' to confirm or 'Escape' to cancel)
```

Exiting the system. Goodbye!

4. Insurance Claim Expert System

Berikut adalah list *rule* pada *expert system* berbasis *Ripple Down Rules* dalam bidang klaim asuransi:

1. collision, at-fault > requires inspection
2. collision, not at-fault > check third-party insurance
3. collision, no police report > request police report before processing
4. collision, at-fault, hit-and-run > apply uninsured motorist coverage
5. damage from animal > apply comprehensive coverage
6. damage from animal, parked vehicle > apply comprehensive coverage
7. hit-and-run, uninsured third party > apply uninsured motorist coverage
8. late claim filing > deny claim
9. late claim filing, theft > request explanation for delay
10. comprehensive > total loss evaluation
11. comprehensive, weather-related > total loss evaluation
12. comprehensive, theft, no police report > deny claim
13. claim exceeds \$15000, medical expenses > verify medical coverage and adjust payout
14. claim exceeds \$15000, multiple injuries, at-fault > flag for supervisor review
15. claim exceeds \$15000, hit-and-run, uninsured motorist > flag for supervisor review
16. luxury vehicle > adjust payout for luxury vehicle standards
17. luxury vehicle, major damage > send for specialized adjuster review
18. luxury vehicle, stolen > verify police report and apply theft coverage
19. electric vehicle > adjust payout for ev repair costs
20. electric vehicle, battery damage > apply comprehensive coverage for battery replacement
21. electric vehicle, fire-related damage > adjust payout for ev-specific parts
22. vehicle total loss > adjust payout for aftermarket parts
23. vehicle total loss, modified aftermarket parts > adjust payout for aftermarket parts
24. vehicle total loss, flood damage > verify flood coverage endorsement
25. pre-existing damage > exclude pre-existing damage from payout
26. policy exclusion > deny claim
27. policy exclusion, racing-related accident > deny claim
28. driver intoxicated > deny claim

29. policy exclusion, driver intoxicated > deny claim
30. suspicious claim > flag for fraud review or request additional documents
31. suspicious claim, inflated repair costs > request secondary inspection
32. driver under 18 > adjust payout based on minor driver rules
33. collision, not at-fault, major damage > check third-party insurance
34. collision, not at-fault, minor damage > process claim under own coverage
35. collision, at-fault, major damage > requires inspection
36. collision, at-fault, minor damage > apply repair estimate
37. damage during off-road use > apply off-road coverage
38. damage during off-road use, no off-road coverage > deny claim
39. damage during off-road use, pre-existing damage > exclude pre-existing damage
40. hit-and-run, uninsured third party, major damage > apply uninsured motorist coverage
41. hit-and-run, uninsured third party, minor damage > apply uninsured motorist coverage for repairs
42. claim filed over 60 days > flag for fraud review
43. claim filed over 60 days, comprehensive > request explanation for late filing
44. claim filed over 60 days, multiple vehicles, at-fault, major damage > adjust payout for each vehicle involved
45. claim filed over 60 days, multiple vehicles, not at-fault, major damage > apply third-party insurance for each vehicle
46. claim exceeds \$15,000, medical expenses > flag for supervisor review
47. theft, vehicle recovered with damage > apply comprehensive coverage for repairs
48. collision, uninsured motorist, at-fault > deny uninsured motorist coverage
49. flood damage > verify flood coverage endorsement
50. flood damage, uninsured third party > deny claim under third-party coverage

Adapun kondisi unik yang diperhitungkan adalah:

1. collision
2. at-fault
3. not at-fault
4. no police report

5. hit-and-run
6. damage from animal
7. parked vehicle
8. uninsured third party
9. late claim filing
10. theft
11. comprehensive
12. weather-related
13. claim exceeds \$15000
14. medical expenses
15. multiple injuries
16. uninsured motorist
17. luxury vehicle
18. major damage
19. stolen
20. electric vehicle
21. battery damage
22. fire-related damage
23. vehicle total loss
24. modified aftermarket parts
25. flood damage
26. pre-existing damage
27. policy exclusion
28. racing-related accident
29. driver intoxicated
30. suspicious claim
31. inflated repair costs
32. driver under 18
33. minor damage
34. damage during off-road use
35. no off-road coverage
36. pre-existing damage
37. claim filed over 60 days

38. multiple vehicles
39. medical expenses
40. vehicle recovered with damage

Berdasarkan kondisi *rule* yang ada, berikut merupakan hasil pembentukan pohon yang didapat:

```
[ 'collision', 'at-fault' ] -> requires inspection
|
| - true node: ['hit-and-run'] -> apply uninsured motorist coverage
| |
| | - false node: ['major damage'] -> requires inspection
| | |
| | | - false node: ['minor damage'] -> apply repair estimate
| | |
| | | - false node: ['uninsured motorist'] -> deny uninsured motorist coverage
| |
| - false node: ['collision', 'not at-fault'] -> check third-party insurance
|
| - true node: ['major damage'] -> check third-party insurance
| |
| | - false node: ['minor damage'] -> process claim under own coverage
| |
| - false node: ['collision', 'no police report'] -> request police report before processing
| |
| | - false node: ['damage from animal'] -> apply comprehensive coverage
| |
| | - true node: ['parked vehicle'] -> apply comprehensive coverage
| |
| | - false node: ['hit-and-run', 'uninsured third party'] -> apply uninsured motorist coverage
| |
| | - true node: ['major damage'] -> apply uninsured motorist coverage
| | |
| | | - false node: ['minor damage'] -> apply uninsured motorist coverage for repairs
| |
| - false node: ['late claim filing'] -> deny claim
| |
| | - true node: ['theft'] -> request explanation for delay
| |
| | - false node: ['comprehensive'] -> total loss evaluation
| |
| | - true node: ['weather-related'] -> total loss evaluation
| | |
| | | - false node: ['theft', 'no police report'] -> deny claim
| | |
| | | - false node: ['claim filed over 60 days'] -> request explanation for late filing
| |
| - false node: ['claim exceeds $15000', 'medical expenses'] -> verify medical coverage and adjust payout
| |
| | - false node: ['claim exceeds $15000', 'multiple injuries', 'at-fault'] -> flag for supervisor review
| |
| | - false node: ['claim exceeds $15000', 'hit-and-run', 'uninsured motorist'] -> flag for supervisor review
| |
| | - false node: ['luxury vehicle'] -> adjust payout for luxury vehicle standards
| |
| | - true node: ['major damage'] -> send for specialized adjuster review
| |
| | - false node: ['stolen'] -> verify police report and apply theft coverage
| |
| | - false node: ['electric vehicle'] -> adjust payout for ev repair costs
| |
| - true node: ['battery damage'] -> apply comprehensive coverage for battery replacement
| |
| | - false node: ['fire-related damage'] -> adjust payout for ev-specific parts
| |
| - false node: ['vehicle total loss'] -> adjust payout for aftermarket parts
| |
| | - true node: ['modified aftermarket parts'] -> adjust payout for aftermarket parts
| |
| | - false node: ['flood damage'] -> verify flood coverage endorsement
| |
| - false node: ['pre-existing damage'] -> exclude pre-existing damage from payout
| |
| | - true node: ['damage during off-road use'] -> exclude pre-existing damage
| |
| | - false node: ['policy exclusion'] -> deny claim
| |
| | - true node: ['racing-related accident'] -> deny claim
| |
| | |
| | | - false node: ['driver intoxicated'] -> deny claim
| |
| | - false node: ['driver intoxicated'] -> deny claim
| |
| | - false node: ['suspicious claim'] -> flag for fraud review or request additional documents
| |
| | - true node: ['inflated repair costs'] -> request secondary inspection
| |
| | - false node: ['driver under 18'] -> adjust payout based on minor driver rules
| |
| | - false node: ['damage during off-road use'] -> apply off-road coverage
| |
| | - true node: ['no off-road coverage'] -> deny claim
| |
| - false node: ['claim filed over 60 days'] -> flag for fraud review
| |
| | - true node: ['multiple vehicles', 'at-fault', 'major damage'] -> adjust payout for each vehicle involved
| | |
| | | - false node: ['multiple vehicles', 'not at-fault', 'major damage'] -> apply third-party insurance for each vehicle
| |
| - false node: ['claim exceeds $15', '000', 'medical expenses'] -> flag for supervisor review
| |
| | - false node: ['theft', 'vehicle recovered with damage'] -> apply comprehensive coverage for repairs
| |
| | - false node: ['flood damage'] -> verify flood coverage endorsement
| |
| | - true node: ['uninsured third party'] -> deny claim under third-party coverage
```

Analisis hasil pembentukan pohon RDR:

Berdasarkan hasil pembentukan pohon, kondisi masukan aturan awal pertama kali dilakukan dengan *condition* dan *result* sebagai berikut:

collision, at-fault > requires inspection

Masukan aturan berikutnya untuk pembentukan pohon yaitu adalah sebagai berikut:

collision, not at-fault > check third-party insurance

Berdasarkan kondisi masukan berikutnya, karena aturan kedua tidak sesuai dengan aturan pertama, maka aturan kedua masuk ke dalam kondisi *false node*. Berikutnya pengguna kemudian memasukkan aturan baru sebagai berikut:

collision, no police report > request police report before processing

Kondisi masukan aturan ini tidak sesuai dengan aturan pertama maupun kedua yang telah dibuat sebelumnya sehingga aturan ini masuk ke dalam kondisi *false node* dari cabang aturan kedua. Berikutnya pengguna kemudian memasukkan aturan baru sebagai berikut:

collision, at-fault, hit-and-run > apply uninsured motorist coverage

Berdasarkan masukan aturan ini, karena kondisi awal yaitu *collision* dan *at-fault* sesuai dengan aturan pertama, maka kondisi baru yaitu *hit-and-run* akan masuk ke dalam *true node* dari aturan pertama. Berikutnya pengguna kemudian memasukkan aturan baru sebagai berikut:

damage from animal > apply comprehensive coverage

Karena aturan ini tidak sesuai dengan keempat aturan sebelumnya, maka aturan ini masuk ke dalam *false node* dari percabangan terakhir yaitu pada aturan ke tiga. Aturan yang ada akan terus berlaku hingga semua aturan sudah terpenuhi untuk membentuk pohon RDR.

5. Kesimpulan

Beberapa kesimpulan yang didapat dari implementasi *Ripple Down Rules* dengan menggunakan Jupyter Notebook adalah sebagai berikut:

- a. Implementasi sistem *Ripple Down Rules* bersifat *robust* yang memberikan kemudahan bagi pakar untuk menambahkan aturan baru tanpa menyebabkan konflik atau perubahan besar pada aturan yang sudah ada
- b. Pembentukan pohon *Ripple Down Rules* akan cenderung terpusat pada kondisi *true node* jika banyak aturan yang terpenuhi dan sebaliknya akan terpusat pada kondisi *false node* jika banyak aturan yang tidak memenuhi kondisi.

Lampiran

Link Repository: https://github.com/Gulilil/IF4070_Tugas2