

Spesifikasi Tugas Besar

Matematika Diskrit Graf

Kompilasi FOL dan PySwimp



033 – 049

Arkan-Julian

Contents

Bab I: Pendahuluan	3
Latar Belakang Masalah	3
Tujuan	4
Pembagian Tugas.....	5
Bab II: Desain Knowledge Base.....	5
Daftar Predikat	5
Daftar Fakta	5
Daftar Rules	6
Rantai Penalaran	6
Bab III: Analisis Teoritis	7
Formulasi FOL.....	7
Analisis Unifikasi (MGU)	7
Pembuktian Resolusi	9
Bab IV: Implementasi dan Pengujian	10
prolog_kb.pl	10
Struktur Aplikasi	12
Hasil Uji Inferensi	17
Uji Query Kustom.....	20
Bab V: Analisis Kritis dan Penutup	21
Keterbatasan FOL	21
Rekomendasi Kebijakan	21
Kesimpulan.....	22

Bab I: Pendahuluan

Latar Belakang Masalah

Di era *streaming* musik, *playlist* telah menjadi medium utama bagi pengguna untuk menikmati koleksi lagu. Namun, pengalaman mendengarkan seringkali terganggu oleh transisi yang kasar antar lagu. Hal ini terjadi ketika sebuah lagu yang tenang tiba-tiba diikuti oleh lagu yang sangat energik, sehingga merusak suasana hati dan tujuan dari *playlist* itu sendiri.

Masalah utama yang diangkat dalam laporan ini adalah penyusunan *playlist* yang tidak mulus, yang berasal dari pengurutan lagu yang tidak mempertimbangkan kesamaan fitur audio antar trek yang berurutan. Isu ini relevan karena memengaruhi jutaan pengguna layanan musik digital.

Untuk memahami secara mendalam mengapa sebuah *playlist* musik seringkali terasa tidak memiliki alur yang mulus, dilakukan analisis akar masalah menggunakan diagram Fishbone. Analisis ini mengungkapkan bahwa masalah "loncatan mood musik" tidak disebabkan oleh satu faktor tunggal, melainkan merupakan hasil dari interaksi kompleks antara perilaku pengguna, sifat koleksi lagu, serta keterbatasan platform musik itu sendiri.

Faktor utama dan yang paling signifikan terletak pada Proses Input Manual oleh Pengguna. Seringkali, sebuah *playlist* tidak dibuat dalam satu waktu, melainkan berkembang secara organik seiring berjalaninya waktu. Pengguna cenderung menambahkan lagu secara spontan berdasarkan mood sesaat, dengan tujuan utama untuk memperkaya koleksi, bukan untuk membangun sebuah alur musical yang koheren. Proses ini diperparah oleh tidak adanya alat bantu penyisipan cerdas dari platform; lagu baru umumnya hanya ditambahkan di akhir atau awal daftar, tanpa mempertimbangkan posisi optimalnya.

Faktor kedua adalah Sifat Inheren dari Koleksi Lagu itu sendiri. Sebuah *playlist* seringkali berisi lagu dari genre yang sangat beragam dengan rentang energi dan tempo yang sangat lebar. Keragaman musical yang luas ini secara alami menciptakan tantangan dalam menyusun urutan yang mulus dari awal hingga akhir.

Selanjutnya, Keterbatasan Platform Musik juga turut berkontribusi. Fitur standar seperti mode putar acak (shuffle) secara desain merupakan antitesis dari alur yang teratur. Opsi pengurutan yang tersedia juga tidak relevan untuk menciptakan pengalaman mendengarkan yang harmonis.

Terakhir, Algoritma Internal Platform seringkali tidak dirancang untuk menyelesaikan masalah ini. Fokus utama algoritma biasanya pada penemuan (discovery) musik baru, bukan pada optimisasi urutan *playlist* yang sudah ada. Akibatnya, metrik kemiripan atau rekomendasi yang digunakan cenderung terlalu sederhana untuk menciptakan transisi yang benar-benar mulus.

Berdasarkan analisis tersebut, berikut adalah rangkuman dalam kerangka diagram Fishbone:

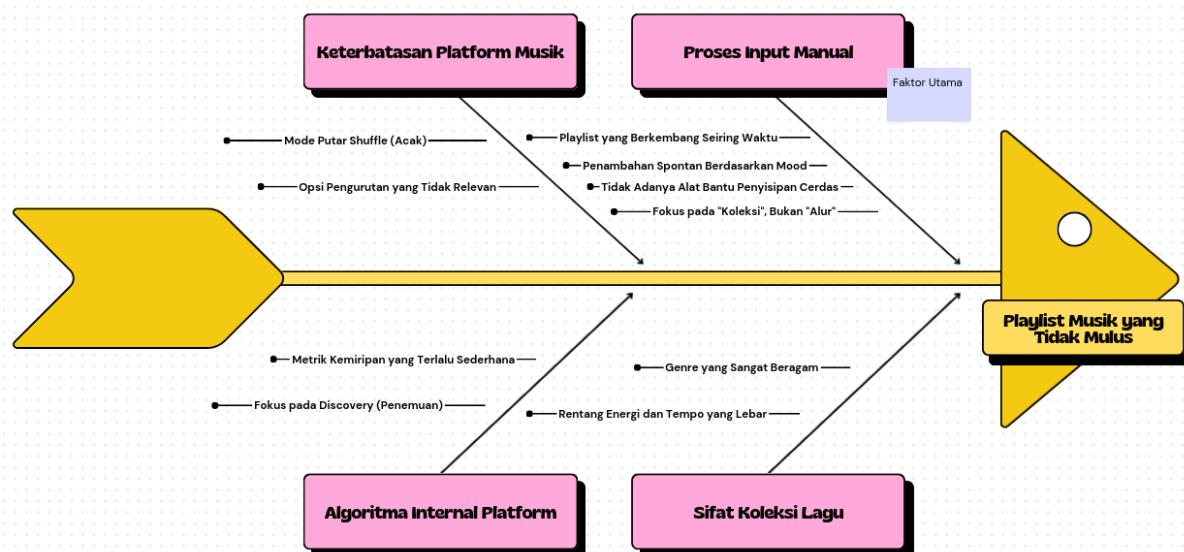
Kepala Ikan:

“*Playlist* Musik yang Tidak Mulus Urutannya”

Tulang ikan dan duri-duri kecil

1. Proses Input Manual Oleh User
 - a. *Playlist* yang berkembang seiring waktu

- b. Penambahan spontan berdasarkan mood user
 - c. Tidak adanya alat bantu penyisipan cerdas
 - d. Fokus pada “Koleksi”, bukan “Alur”
2. Sifat Koleksi Lagu
- a. Genre yang sangat beragam
 - b. Rentang Energi dan Tempo yang Lebar
3. Keterbatasan Platform Musik
- a. Mode Putar Shuffle
 - b. Opsi Pengurutan yang tidak relevan
4. Agoritma Internal Platform
- a. Metrik kemiripan yang terlalu sederhana
 - b. Fokus pada Discovery (penemuan)



Tujuan

Berdasarkan latar belakang masalah di atas, tujuan dari pembangunan sistem inferensi kebijakan ini adalah:

1. Membangun representasi formal (Logika Orde Pertama/FOL) untuk mendefinisikan hubungan antar fitur audio (energi, genre, tempo) guna menetapkan standar transisi lagu yang harmonis (misalnya: mencegah jumpscare audio).
2. Menyusun basis pengetahuan (Knowledge Base) komputasional menggunakan Prolog yang memuat fakta atribut lagu dan aturan inferensi (rules) untuk mendeteksi potensi ketidakselaras dalam urutan playlist.
3. Menguji validitas aturan transisi yang telah dibuat melalui simulasi inferensi otomatis (menggunakan PySwip/Streamlit) untuk membuktikan bahwa penerapan logika kausalitas dapat meminimalkan transisi kasar secara objektif.
4. Menghasilkan rekomendasi kebijakan kurasi otomatis yang berbasis aturan logika (rule-based policy) sebagai solusi alternatif terhadap keterbatasan pengurutan manual dan pengacak (shuffle) pada platform musik saat ini.

Pembagian Tugas

No	Nama	NIM	Rincian Tugas	Persentase Pengerojaan
1	Arkan Ramadhan Nugraha	241524033	Membangun Knowledge Base (Prolog) dan GUI (Streamlit). Merancang rules dan formulasi FOL. Menyusun Laporan	50%
2	Julian Dio Saputra	241524049	Menjalankan skenario uji coba aplikasi untuk keperluan perekaman demo. Menerjemahkan logika program ke dalam bentuk visual (PPT) agar mudah dipahami. <i>Editing</i> video demo dan penyusunan materi presentasi akhir.	50%

Bab II: Desain Knowledge Base

Daftar Predikat

1. memiliki_atribut(<lagu>, <genre>, <tingkat energi>, <kecepatan tempo>)
2. jarak_graf(<lagu1>, <lagu2>, <nilai>)
3. beda_genre(<lagu1>, <lagu2>)
4. energi_kontras(<lagu1>, <lagu2>)
5. transisi_kasar(<lagu1>, <lagu2>)
6. transisi_harmonis(<lagu1>, <lagu2>)
7. butuh_lagu_bridge(<lagu1>, <lagu2>)
8. rekomendasi_urutan(<lagu1>, <lagu2>)

Daftar Fakta

1. lagu(beautiful_world).
2. lagu(good_luck_babe).
3. lagu(russian_roulette).
4. lagu(tripping_wires).
5. lagu(self_care).
6. memiliki_atribut(beautiful_world, shoegaze, tinggi, sedang).
7. memiliki_atribut(good_luck_babe, pop, tinggi, cepat).
8. memiliki_atribut(russian_roulette, electronic, tinggi, cepat).

9. memiliki_atribut(self_care, hip_hop, sedang, sedang).
10. memiliki_atribut(tripping_wires, metal, sangat_tinggi, cepat).
11. jarak_graf(tripping_wires, self_care, 0.95).
12. jarak_graf(tripping_wires, good_luck_babe, 0.8).
13. jarak_graf(good_luck_babe, russian_roulette, 0.2).
14. jarak_graf(beautiful_world, russian_roulette, 0.6).
15. jarak_graf(self_care, good_luck_babe, 0.45).

Daftar Rules

1. lagu_ekstrem(X) :- memiliki_atribut(X, _, sangat_tinggi, _) ; memiliki_atribut(X, _, rendah, _).
2. pembentuk_outlier(X) :- lagu_ekstrem(X).
3. wajib_kurasi_manual(X) :- pembentuk_outlier(X).
4. beda_genre(X, Y) :- memiliki_atribut(X, Genre1, _, _), memiliki_atribut(Y, Genre2, _, _), Genre1 != Genre2.
5. energi_kontras(X, Y) :- memiliki_atribut(X, _, sangat_tinggi, _), (memiliki_atribut(Y, _, sedang, _)) ; memiliki_atribut(Y, _, rendah, _)).
6. transisi_kasar(X, Y) :- energi_kontras(X, Y) ; (jarak_graf(X, Y, Jarak), Jarak > 0.7).
7. transisi_harmonis(X, Y) :- jarak_graf(X, Y, Jarak), Jarak <= 0.4.
8. butuh_lagu_bridge(X, Y) :- transisi_kasar(X, Y).
9. rekomendasi_urutan(X, Y) :- transisi_harmonis(X, Y).

Rantai Penalaran

Berikut adalah satu contoh simulasi penalaran rantai 3 langkah:

butuh_lagu_bridge(tripping_wires, self_care)

1. Cari fakta
 - a. Sistem mengecek memiliki_atribut(tripping_wires, ..., sangat_tinggi, ...)
 - b. Sistem mengecek memiliki_atribut(self_care, ..., sedang, ...)
2. Evaluasi Rule 6: energi_kontras(X, Y) :- memiliki_atribut(X, _, sangat_tinggi, _), memiliki_atribut(Y, _, sedang, _).
 - a. Karena tripping_wires memiliki tingkat energi sangat_tinggi dan self_care memiliki tingkat energi sedang, maka kondisi Rule 6 terpenuhi
 - b. energi_kontras(tripping_wires, self_care) bernilai true
3. Evaluasi Rule 8: transisi_kasar(X, Y) :- energi_kontras(X, Y).
 - a. Karena energi_kontras benilai true, maka Rule 8 terpenuhi
 - b. transisi_kasar(tripping_wires, self_care) bernilai true
4. Evaluasi Rule 11: butuh_lagu_bridge(X, Y) :- transisi_kasar(X, Y).
 - a. Karena transisi_kasar bernilai true, maka rule 11 terpenuhi
5. Hasil akhir
 - a. butuh_lagu_bridge(tripping_wires, self_care) bernilai true
 - b. Rantai 3 langkah

$$\begin{aligned} \forall X \forall Y ((\text{memiliki_atribut}(X, g_1, \text{sangat_tinggi}, t_1) \\ \wedge (\text{memiliki_atribut}(Y, g_2, \text{sedang}, t_2) \\ \vee \text{memiliki_atribut}(Y, g_3, \text{rendah}, t_3))) \rightarrow \text{energi_kontras}(X, Y)) \\ \forall X \forall Y \forall N ((\text{energi_kontras}(X, Y) \vee (\text{jarak_graf}(X, Y, N) \wedge (N > 0.7))) \\ \rightarrow \text{transisi_kasar}(X, Y)) \end{aligned}$$

$$\forall X \forall Y (\text{transisi_kasar}(X, Y) \rightarrow \text{butuh_lagu_bridge}(X, Y))$$

Bab III: Analisis Teoritis

Formulasi FOL

1. $\text{lagu_ekstrem}(X) :- \text{memiliki_atribut}(X, _, \text{sangat_tinggi}, _) ; \text{memiliki_atribut}(X, _, \text{rendah}, _).$
$$\forall X ((\text{memiliki_atribut}(X, g_1, \text{sangat_tinggi}, t_1) \vee \text{memiliki_atribut}(X, g_2, \text{rendah}, t_2)) \rightarrow \text{lagu_ekstrem}(X))$$
2. $\text{pembentuk_outlier}(X) :- \text{lagu_ekstrem}(X).$
$$\forall X (\text{lagu_ekstrem}(X) \rightarrow \text{pembentuk_outlier}(X))$$
3. $\text{wajib_kurasi_manual}(X) :- \text{pembentuk_outlier}(X).$
$$\forall X (\text{pembentuk_outlier}(X) \rightarrow \text{wajib_kurasi_manual}(X))$$
4. $\text{beda_genre}(X, Y) :- \text{memiliki_atribut}(X, \text{Genre1}, _, _), \text{memiliki_atribut}(Y, \text{Genre2}, _, _), \text{Genre1} \neq \text{Genre2}.$
$$\forall X \forall Y ((\text{memiliki_atribut}(X, g_1, e_1, t_1) \wedge \text{memiliki_atribut}(Y, g_2, e_2, t_2) \wedge (g_1 \neq g_2)) \rightarrow \text{beda_genre}(X, Y))$$
5. $\text{energi_kontras}(X, Y) :- \text{memiliki_atribut}(X, _, \text{sangat_tinggi}, _), (\text{memiliki_atribut}(Y, _, \text{sedang}, _) ; \text{memiliki_atribut}(Y, _, \text{rendah}, _)).$
$$\forall X \forall Y ((\text{memiliki_atribut}(X, g_1, \text{sangat_tinggi}, t_1) \wedge (\text{memiliki_atribut}(Y, g_2, \text{sedang}, t_2) \vee \text{memiliki_atribut}(Y, g_3, \text{rendah}, t_3))) \rightarrow \text{energi_kontras}(X, Y))$$
6. $\text{transisi_kasar}(X, Y) :- \text{energi_kontras}(X, Y) ; (\text{jarak_graf}(X, Y, \text{Jarak}), \text{Jarak} > 0.7).$
$$\forall X \forall Y \forall N ((\text{energi_kontras}(X, Y) \vee (\text{jarak_graf}(X, Y, N) \wedge (N > 0.7))) \rightarrow \text{transisi_kasar}(X, Y))$$
7. $\text{transisi_harmonis}(X, Y) :- \text{jarak_graf}(X, Y, \text{Jarak}), \text{Jarak} \leq 0.4.$
$$\forall X \forall Y \forall N ((\text{jarak_graf}(X, Y, N) \wedge (N \leq 0.4)) \rightarrow \text{transisi_harmonis}(X, Y))$$
8. $\text{butuh_lagu_bridge}(X, Y) :- \text{transisi_kasar}(X, Y).$
$$\forall X \forall Y (\text{transisi_kasar}(X, Y) \rightarrow \text{butuh_lagu_bridge}(X, Y))$$
9. $\text{rekomendasi_urutan}(X, Y) :- \text{transisi_harmonis}(X, Y).$
$$\forall X \forall Y (\text{transisi_harmonis}(X, Y) \rightarrow \text{rekomendasi_urutan}(X, Y))$$

Analisis Unifikasi (MGU)

Analisis apa yang terjadi ketika Prolog diberi query berikut:

Query: ?- `pembentuk_outlier(tripping_wires)`

1. Langkah 1: Unifikasi Goal Awal dengan Rule Tertinggi. Prolog mencoba mencocokkan Query dengan Head dari Rule 2.
 - Goal: `pembentuk_outlier(tripping_wires)`
 - Rule Head (R2): `pembentuk_outlier(X)`
 - Proses Unifikasi
 - Nama predikat sama: `pembentuk_outlier`

- Argumen dicocokkan: Konstanta tripping_wires dicocokkan dengan Variable X
 - MGU: { X = tripping_wires }
 - Status: Berhasil
 - Konsekuensi: Body dari Rule 2 menjadi Goal baru, yaitu lagu_ekstrem(X)
- 2. Langkah 2: Unifikasi Body dengan Rule Dasar. Prolog mencoba membuktikan goal baru lagi dengan mencocokkannya ke Head dari Rule 1
 - Goal: lagu_ekstrem(tripping_wires)
 - Rule Head (R1): lagu_ekstrem(X)
 - Proses Unifikasi
 - Nama predikat sama
 - Argumen dicocokkan: Konstanta tripping_wires dicocokkan dengan variable X pada Rule 2
 - MGU: { X = tripping_wires}
 - Status berhasil
- 3. Konsekuensi: Body dari Rule 1 menjadi Goal terakhir. Karena ada operator OR (;), Prolog mengecek cabang pertama: memiliki_atribut(tripping_wires, _, sangat_tinggi, _)Unifikasi dengan Facts. Prolog mencari fakta di Knowledge Base yang cocok dengan goal terakhir
 - Current Goal: memiliki_atribut(tripping_wires, _, sangat_tinggi, _)
 - Fact di KB: memiliki_atribut(tripping_wires, metal, sangat_tinggi, cepat)
 - Proses Unifikasi:
 - Predikat sama
 - Argumen 1: tripping_wires == tripping_wires
 - Argumen 2: _ (Anonymous Variable) cocok dengan metal.
 - Argumen 3: sangat_tinggi == sangat_tinggi
 - Argumen 4: _ (Anonymous Variable) cocok dengan cepat.
 - MGU: {}
 - Himpunan kosong, karena tidak ada variabel baru yang perlu diikat nilainya untuk pengguna, semua konstanta sudah cocok
 - Status: berhasil

Karena unifikasi berhasil hingga ke fakta dasar, maka Prolog merambatkan nilai True Kembali ke atas, sehingga Query awal pembentuk_outlier(tripping_wires) terbukti valid.

Pembuktian Resolusi

Konversi ke Clausal Form (CNF)

Rule 1

$$\begin{aligned} \forall X & \left((\text{memiliki_atribut}(X, g_1, \text{sangat_tinggi}, t_1) \vee \text{memiliki_atribut}(X, g_2, \text{rendah}, t_2)) \right. \\ & \quad \rightarrow \text{lagu_ekstrem}(X) \Big) \\ & (\neg \text{memiliki_atribut}(X, g_1, \text{sangat_tinggi}, t_1) \wedge \neg \text{memiliki_atribut}(X, g_2, \text{rendah}, t_2)) \\ & \quad \vee \text{lagu_ekstrem}(X) \end{aligned}$$

- Klausus 1.1:
 $\neg \text{memiliki_atribut}(X, \dots, \text{sangat_tinggi}, \dots) \vee \text{lagu_ekstrem}(X)$
- Klausus 1.2:
 $\neg \text{memiliki_atribut}(X, \dots, \text{rendah}, \dots) \vee \text{lagu_ekstrem}(X)$

Rule 2

$$\forall X (\text{lagu_ekstrem}(X) \rightarrow \text{pembentuk_outlier}(X))$$

- Klausus 2:
 $\neg \text{lagu_ekstrem}(X) \vee \text{pembentuk_outlier}(X)$

Fact

$\text{memiliki_atribut}(\text{tripping_wires}, \text{metal}, \text{sangat_tinggi}, \text{cepat})$

Pembuktian Resolusi (Proof by Contradiction). Klausus 3:

$$\neg C: \neg \text{pembentuk_outlier}(\text{tripping_wires})$$

1. Resolusi 1: Gabungkan negated goal dengan Klausus 2
 - a. Unifikasi: Substitusi { X = *tripping_wires* }
 - b. Eliminasi: pembentuk_outlier saling menghilangkan, karena satu negative satu positif
 - c. Hasil

$$\neg \text{lagu_ekstrem}(\text{tripping_wires})$$

2. Resolusi 2: Gabungkan Hasil awal dengan Klausus 1.1 ita memilih Klausus 1.1 karena fakta yang tersedia di KB memiliki atribut "sangat_tinggi". Klausus 1.2 ("rendah") tidak relevan dengan fakta *tripping_wires*.
 - a. Unifikasi: Substitusi { X = *tripping_wires* }

- b. Eliminasi lagu_ekstrem. Saling menghilangkan
 - c. Hasil
- $\neg \text{memiliki_atribut}(\text{tripping_wires}, g_1, \text{sangat_tinggi}, t_1)$

3. Resolusi 3: Gabungkan Hasil kedua dengan Fact

- a. Eliminasi: keduanya Adalah predikat yang identic tetapi berlawanan tanda, sehingga langsung saja elminasi
- b. Hasil akhir

\emptyset

Karena kita menemukan Kontradiksi, maka Asumsi Negasi (Langkah a) adalah salah. Oleh karena itu, Goal Awal (pembentuk_outlier(tripping_wires)) Terbukti benar (Valid).

Bab IV: Implementasi dan Pengujian

prolog_kb.pl

```
% =====
% FILE: prolog_kb.pl
% Knowledge Base Logika Orde Pertama (FOL) untuk Analisis Isu Kontemporer
% Topik: Optimasi Urutan Playlist untuk Transisi Mulus Menggunakan Model Graf
Berbobot
% =====

% --- TEMA: Optimasi Urutan Playlist untuk Transisi Mulus Menggunakan Model Graf
Berbobot ---
% Predikat Utama: lagu, memiliki_atribut, jarak_graf, lagu_ekstrem,
pembentuk_outlier, transisi_kasar, butuh_lagu_bridge

% -----
% 1. DEFINISI FACTS (Minimal 15 Facts / Proposisi Dasar)
% Format: lagu(Nama).
%         memiliki_atribut(Nama, Genre, Energi, Tempo).
%         jarak_graf(Lagu1, Lagu2, Weight).
% -----


% --- Fakta 1-5: Definisi Entitas Lagu ---
% Format: lagu(Nama).
lagu(beautiful_world).
lagu(good_luck_babe).
lagu(russian_roulette).
lagu(tripping_wires).
lagu(self_care).

% --- Fakta 6-10: Atribut Lagu (Genre, Energi, Tempo) ---
% Format: memiliki_atribut(Lagu, Genre, Energi, Tempo)
memiliki_atribut(beautiful_world, shoegaze, tinggi, sedang).
memiliki_atribut(good_luck_babe, pop, tinggi, cepat).
memiliki_atribut(russian_roulette, electronic, tinggi, cepat).
```

```

memiliki_atribut(self_care, hip_hop, sedang, sedang).
memiliki_atribut(tripping_wires, metal, sangat_tinggi, cepat).

% --- Fakta 11-15: Jarak Graf (Bobot Ketidakmiripan) ---
% Semakin tinggi nilai, semakin tidak cocok (jauh)
% Format: jarak_graf(Lagu1, Lagu2, Weight)
jarak_graf(tripping_wires, self_care, 0.95).
jarak_graf(tripping_wires, good_luck_babe, 0.8).
jarak_graf(good_luck_babe, russian_roulette, 0.2).
jarak_graf(beautiful_world, russian_roulette, 0.6).
jarak_graf(self_care, good_luck_babe, 0.45).

% -----
% 2. DEFINISI RULES (Minimal 8 Rules / Implikasi FOL)
% Variabel diawali huruf KAPITAL (X, Y, N).
% -----

% --- BAGIAN 1: RANTAI INFERENSI 3 LANGKAH (Analisis Outlier/Node) ---

% Rule 1 (Premis 1): Identifikasi Lagu Ekstrem
% FOL:  $\forall X ((\text{energi}(X, \text{sangat\_tinggi}) \vee \text{energi}(X, \text{rendah})) \rightarrow \text{lagu\_ekstrem}(X))$ 
lagu_ekstrem(X) :-
    memiliki_atribut(X, _, sangat_tinggi, _);
    memiliki_atribut(X, _, rendah, _).

% Rule 2 (Premis 2): Lagu ekstrem berpotensi menjadi outlier
% FOL:  $\forall X (\text{lagu\_ekstrem}(X) \rightarrow \text{pembentuk\_outlier}(X))$ 
pembentuk_outlier(X) :-
    lagu_ekstrem(X).

% Rule 3 (Premis 3): Outlier wajib dikurasi manual (tidak boleh auto-shuffle)
% FOL:  $\forall X (\text{pembentuk\_outlier}(X) \rightarrow \text{wajib\_kurasi\_manual}(X))$ 
wajib_kurasi_manual(X) :-
    pembentuk_outlier(X).

% --- BAGIAN 2: LOGIKA HUBUNGAN ANTAR LAGU (Analisis Edge/Transisi) ---

% Rule 4: Pengecekan Beda Genre
% FOL:  $\forall X, Y (\text{genre}(X) \neq \text{genre}(Y) \rightarrow \text{beda\_genre}(X, Y))$ 
beda_genre(X, Y) :-
    memiliki_atribut(X, Genre1, _, _),
    memiliki_atribut(Y, Genre2, _, _),
    Genre1 \= Genre2.

% Rule 5: Energi Kontras (Aturan Khusus)
% Jika X sangat tinggi, dan Y sedang atau rendah -> Kontras
% FOL:  $\forall X, Y (\text{sangat\_tinggi}(X) \wedge (\text{sedang}(Y) \vee \text{rendah}(Y)) \rightarrow \text{energi\_kontras}(X, Y))$ 
energi_kontras(X, Y) :-
    memiliki_atribut(X, _, sangat_tinggi, _),
    (
        memiliki_atribut(Y, _, sedang, _);
        memiliki_atribut(Y, _, rendah, _)
    ).

% Rule 6: Transisi Kasar (Penyebab utama butuh bridge)
% Terjadi jika energi kontras ATAU jarak graf > 0.7
% FOL:  $\forall X, Y (\text{energi\_kontras}(X, Y) \vee (\text{jarak}(X, Y) > 0.7) \rightarrow \text{transisi\_kasar}(X, Y))$ 
transisi_kasar(X, Y) :-
    energi_kontras(X, Y);
    jarak_graf(X, Y, Jarak), Jarak > 0.7.

```

```

% Rule 7: Transisi Harmonis (Ideal untuk rekomendasi)
% Terjadi jika jarak graf <= 0.4
% FOL:  $\forall X,Y \text{ (jarak}(X,Y) \leq 0.4 \rightarrow \text{transisi\_harmonis}(X,Y))$ 
transisi_harmonis(X, Y) :-
    jarak_graf(X, Y, Jarak),
    Jarak =< 0.4.

% Rule 8: Kesimpulan Butuh Bridge
% FOL:  $\forall X,Y \text{ (transisi\_kasar}(X,Y) \rightarrow \text{butuh\_lagu\_bridge}(X,Y))$ 
butuh_lagu_bridge(X, Y) :-
    transisi_kasar(X, Y).

% Rule 9: Kesimpulan Rekomendasi Urutan
% FOL:  $\forall X,Y \text{ (transisi\_harmonis}(X,Y) \rightarrow \text{rekomendasi\_urutan}(X,Y))$ 
rekomendasi_urutan(X, Y) :-
    transisi_harmonis(X, Y).

% =====
% END OF FILE
% =====

```

Struktur Aplikasi

Implementasi antarmuka pengguna (GUI) dibangun menggunakan framework Streamlit. Struktur aplikasi ini terdiri dari tiga komponen utama:

1. Inisialisasi Knowledge Base
Menghubungkan Python dengan logic engine SWI-Prolog dan memuat file prolog_kb.pl.
2. Definisi Inferensi
Menyusun daftar 8 rule utama ke dalam struktur data (dictionary) agar dapat dipanggil secara dinamis.
3. Visualisasi Hasil
Mengonversi hasil query dari Prolog (yang berupa raw JSON/List) menjadi Dataframe Pandas agar tampil rapi dalam bentuk tabel di antarmuka web.

```

# =====
# app_inferensi_kebijakan.py
# =====
# Aplikasi GUI berbasis Streamlit untuk:
# - Menjalankan inferensi logika (First Order Logic)
# - Mengintegrasikan Python dengan SWI-Prolog
# - Menguji rule-based reasoning pada optimasi playlist musik
#
# Teknologi:
# - Python (GUI & kontrol alur)
# - Streamlit (User Interface)
# - SWI-Prolog + pyswip (Knowledge Base & Inferensi)
#
# Tujuan akademik:
# - Demonstrasri reasoning berbasis aturan (rule-based system)
# - Menunjukkan chaining inferensi logika
# - Visualisasi hasil query Prolog dalam bentuk tabel
# =====

# =====
# IMPORT LIBRARY
# =====
# pandas
# Digunakan untuk mengubah hasil query Prolog (list of dict)

```

```

# menjadi DataFrame agar mudah ditampilkan dalam GUI
import pandas as pd

# streamlit
# Framework GUI berbasis web untuk Python
# Digunakan untuk:
# - Layout halaman
# - Tombol
# - Tabel
# - Expander
# - Input user
import streamlit as st

# pyswip.Prolog
# Library Python untuk berinteraksi dengan SWI-Prolog
# Digunakan untuk:
# - Load knowledge base (.pl)
# - Menjalankan query Prolog
from pyswip import Prolog

# =====
# 1. SETUP APLIKASI & LOAD KNOWLEDGE BASE
# =====

# Nama file Knowledge Base Prolog
# Berisi fakta dan aturan inferensi playlist
KB_FILE = "prolog_kb.pl"

# Konfigurasi halaman Streamlit
# layout="wide" agar tampilan lebih lebar dan nyaman
st.set_page_config(layout="wide")

# Judul utama aplikasi
st.title("🎵 Logic Programming GUI: Optimasi Urutan Playlist Musik")

# =====
# Inisialisasi Engine Prolog
# =====

# Menggunakan st.session_state agar:
# - Prolog hanya di-load sekali
# - Tidak reload setiap interaksi tombol
# - Lebih efisien
if "prolog" not in st.session_state:
    try:
        # Membuat instance Prolog
        prolog = Prolog()

        # Load knowledge base (.pl)
        prolog.consult(KB_FILE)

        # Simpan ke session_state
        st.session_state.prolog = prolog
        st.session_state.kb_loaded = True

    except Exception as e:
        # Jika gagal load KB:
        # - tampilkan error
        # - hentikan aplikasi
        st.error("Gagal memuat SWI-Prolog atau Knowledge Base.")
        st.error(e)
        st.stop()

# Ambil instance Prolog dari session

```

```

prolog = st.session_state.prolog

# Notifikasi sukses load KB
st.success(f"Knowledge Base '{KB_FILE}' berhasil dimuat")

# =====
# 2. DAFTAR INFERENSI (QUERY WAJIB)
# =====
# Setiap inferensi merepresentasikan:
# - Satu rule atau kesimpulan logika
# - Query Prolog
# - Penjelasan konseptual
#
# Inferensi ini digunakan untuk menunjukkan:
# - Rule dasar
# - Chaining inferensi
# - Reasoning bertingkat
inferensi_list = [
    {
        "nama": "Inferensi 1: Lagu Ekstrem",
        "query": "lagu_ekstrem(X)",
        "deskripsi": "Rule 1 - Lagu dengan energi sangat tinggi atau rendah",
    },
    {
        "nama": "Inferensi 2: Pembentuk Outlier",
        "query": "pembentuk_outlier(X)",
        "deskripsi": "Rule 2 - Lagu ekstrem berpotensi menjadi outlier",
    },
    {
        "nama": "Inferensi 3: Wajib Kurasi Manual",
        "query": "wajib_kurasi_manual(X)",
        "deskripsi": "Rule 3 - Rantai inferensi (ekstrem → outlier → kurasi)",
    },
    {
        "nama": "Inferensi 4: Energi Kontras",
        "query": "energi_kontras(X, Y)",
        "deskripsi": "Rule 5 - Perbedaan energi lagu signifikan",
    },
    {
        "nama": "Inferensi 5: Transisi Kasar",
        "query": "transisi_kasar(X, Y)",
        "deskripsi": "Rule 6 - Energi kontras atau jarak graf besar",
    },
    {
        "nama": "Inferensi 6: Butuh Lagu Bridge",
        "query": "butuh_lagu_bridge(X, Y)",
        "deskripsi": "Rule 8 - Kesimpulan dari transisi kasar",
    },
    {
        "nama": "Inferensi 7: Transisi Harmonis",
        "query": "transisi_harmonis(X, Y)",
        "deskripsi": "Rule 7 - Jarak graf kecil (transisi halus)",
    },
    {
        "nama": "Inferensi 8: Rekomendasi Urutan Playlist",
        "query": "rekомendasi_urutan(X, Y)",
        "deskripsi": "Rule 9 - Pasangan lagu ideal",
    },
]

# =====
# 3. FUNGSI EKSEKUSI QUERY PROLOG

```

```

# =====
def run_query(query):
    """
        Menjalankan query Prolog dan menginterpretasikan hasilnya.

        Kemungkinan hasil:
        1. False → tidak ada solusi
        2. True → query valid tanpa variabel
        3. Binding variabel → ditampilkan sebagai tabel
    """
    try:
        # Jalankan query Prolog
        results = list(prolog.query(query))

        # Jika tidak ada hasil → False
        if not results:
            return "TIDAK VALID (False)"

        # Jika hanya True tanpa binding variabel
        if len(results) == 1 and results[0] == {}:
            return "VALID (True)"

        # Jika ada binding variabel
        # Konversi ke list of dict agar bisa jadi DataFrame
        table = []
        for res in results:
            row = {k: str(v) for k, v in res.items()}
            table.append(row)

        return table

    except Exception as e:
        # Tangani error query Prolog
        return f"ERROR Query Prolog: {e}"

# =====
# 4. TAMPILAN GUI
# =====

# Layout dua kolom:
# - Kiri : Knowledge Base
# - Kanan : Inferensi & Query
col_kb, col_query = st.columns([1, 2])

# =====
# KOLOM KIRI - KNOWLEDGE BASE
# =====
with col_kb:
    st.header("��识库 Prolog")

    # Tampilkan isi file Prolog
    try:
        with open(KB_FILE, "r") as f:
            kb_content = f.read()
        st.code(kb_content, language="prolog")
    except FileNotFoundError:
        st.warning("File KB tidak ditemukan.")

# =====
# KOLOM KANAN - INFERENCE & QUERY
# =====
with col_query:

```

```

st.header("⌚ Uji Inferensi Logika (FOL)")
st.info("Klik tombol untuk menjalankan inferensi dari Knowledge Base")

# Loop setiap inferensi
for i, item in enumerate(inferensi_list):
    with st.expander(item["nama"], expanded=False):
        st.markdown(f"**Tujuan:** {item['deskripsi']}")
        st.code(item["query"], language="prolog")

    # Tombol eksekusi inferensi
    if st.button(f"UJI INFERENCE {i + 1}", key=f"btn_{i}"):
        result = run_query(item["query"])

        if isinstance(result, list):
            st.success(f"Ditemukan {len(result)} hasil:")
            st.dataframe(pd.DataFrame(result), use_container_width=True)
        else:
            st.info(result)

# =====
# QUERY KUSTOM
# =====
st.markdown("---")
st.subheader("🔍 Query Prolog Kustom")

# Input query bebas dari user
custom_query = st.text_input("Masukkan Query Prolog (contoh: beda_genre(X, Y).)")

# Tombol eksekusi query kustom
if st.button("JALANKAN QUERY KUSTOM"):
    if custom_query.strip():
        result = run_query(custom_query)

        if isinstance(result, list):
            st.success("Hasil Query Kustom:")
            st.dataframe(pd.DataFrame(result), use_container_width=True)
        else:
            st.info(result)
    else:
        st.warning("Query tidak boleh kosong.")

```

Knowledge Base Prolog

```
% *****%
% FILE: prolog_kb.pl
% Knowledge Base Logika Orde Pertama (FOL) untuk Analisis Jsu Kon
% *****%
%
% --- TEMA: Optimasi Urutan Playlist untuk Transisi Mulus Menggunakan Predikat Utama: lagu, memiliki_atribut, jarak_graf, lagu_ekstrem.
%
% 1. DEFINISI FACTS (Minimal 8 Facts / Proposisi Dasar)
% Format: lagu(Nama),
% memiliki_atribut(Nama, Genre, Energi, Tempo).
% jarak_graf(Lagu1, Lagu2, Weight).
%
% ---- Fakta 1-5: Definisi Entitas Lagu ---
lagu(beautiful_world).
lagu(good_luck_babe).
lagu(russian_roulette).
lagu(tripping_wires).
lagu(self_care).

% ---- Fakta 6-10: Atribut Lagu (Genre, Energi, Tempo) ---
% Format: memiliki_atribut(Lagu, Genre, Energi, Tempo)
memiliki_atribut(beautiful_world, shoegaze, tinggi, cepat).
memiliki_atribut(good_luck_babe, pop, tinggi, cepat).
memiliki_atribut(russian_roulette, electronic, tinggi, cepat).
memiliki_atribut(self_care, hip_hop, sedang, sedang).
memiliki_atribut(tripping_wires, metal, sangat_tinggi, cepat).

% ---- Fakta 11-15: Jarak Graf (Bobot Ketidakteripon) ---
% Semakin tinggi nilainya, semakin tidak cocok (jauh)
```

Uji Inferensi Logika (FOL)

Klik tombol untuk menjalankan inferensi dari Knowledge Base

- > Inferensi 1: Lagu Ekstrem
- > Inferensi 2: Pembentuk Outlier
- > Inferensi 3: Wajib Kurasi Manual
- > Inferensi 4: Energi Kontras
- > Inferensi 5: Transisi Kasar
- > Inferensi 6: Butuh Lagu Bridge
- > Inferensi 7: Transisi Harmonis
- > Inferensi 8: Rekomendasi Urutan Playlist

Query Prolog Kustom

Masukkan Query Prolog (contoh: beds_genre(X, Y).)

Hasil Uji Inferensi

Pengujian dilakukan dengan menekan tombol yang tersedia di GUI

Hasil eksekusi:

1. Inferensi 1: Lagu Ekstrem

Inferensi 1: Lagu Ekstrem

Tujuan: Rule 1 – Lagu dengan energi sangat tinggi atau rendah

lagu_ekstrem(X)

UJI INFERENSI 1

Ditemukan 1 hasil:

X
tripping_wires

Sistem berhasil mengidentifikasi `tripping_wires` sebagai lagu ekstrem karena memenuhi aturan atribut energi sangat_tinggi. Hasil ditampilkan dalam format tabel (variabel X)

2. Inferensi 2: Pembentuk Outlier

▼ Inferensi 2: Pembentuk Outlier

Tujuan: Rule 2 – Lagu ekstrem berpotensi menjadi outlier

`pembentuk_outlier(X)`

UJI INFERENSI 2

Ditemukan 1 hasil:

X	tripping_wires
0	

Sistem berhasil mengidentifikasi `tripping_wires` sebagai `pembentuk_outlier` karena `lagu_ekstrem(tripping_wires)`

3. Inferensi 3: Wajib Kurasi Manual

▼ Inferensi 3: Wajib Kurasi Manual

Tujuan: Rule 3 – Rantai inferensi 3 langkah (ekstrem → outlier → kurasi)

`wajib_kurasi_manual(X)`

UJI INFERENSI 3

Ditemukan 1 hasil:

X	tripping_wires
0	

Sistem berhasil mengidentifikasi `tripping_wires` sebagai `wajib_kurasi_manual` karena `pembentuk_outlier(tripping_wires)`

4. Inferensi 4: Energi Kontras

▼ Inferensi 4: Energi Kontras

Tujuan: Rule 5 – Lagu energi sangat tinggi ke sedang/rendah

`energi_kontras(X, Y)`

UJI INFERENSI 4

Ditemukan 1 hasil:

X	Y
tripping_wires	self_care
0	

Sistem berhasil menampilkan pasangan lagu (X, Y) yang mempunyai energi yang kontras.

5. Inferensi 5: Transisi Kasar

▼ Inferensi 5: Transisi Kasar

Tujuan: Rule 6 – Energi kontras atau jarak graf > 0.7

```
transisi_kasar(X, Y)
```

UJI INFERENSI 5

Ditemukan 3 hasil:

	X	Y
0	tripping_wires	self_care
1	tripping_wires	self_care
2	tripping_wires	good_luck_babe

Sistem berhasil menampilkan pasangan lagu (X, Y) yang mempunyai transisi kasar, disebabkan oleh energi yang kontras atau jarak graf yang lebih dari 0.7

6. Inferensi 6: Butuh Lagu Bridge

▼ Inferensi 6: Butuh Lagu Bridge

Tujuan: Rule 8 – Kesimpulan dari transisi kasar

```
butuh_lagu_bridge(X, Y)
```

UJI INFERENSI 6

Ditemukan 3 hasil:

	X	Y
0	tripping_wires	self_care
1	tripping_wires	self_care
2	tripping_wires	good_luck_babe

Sistem berhasil menampilkan pasangan lagu (X, Y) yang membutuhkan bridge. Output ini membuktikan bahwa rule transisi_kasar berhasil memicu butuh_lagu_bridge

7. Inferensi 7: Transisi Harmonis

▼ Inferensi 7: Transisi Harmonis

Tujuan: Rule 7 – Jarak graf ≤ 0.4

```
transisi_harmonis(X, Y)
```

UJI INFERENSI 7

Ditemukan 1 hasil:

	X	Y
0	good_luck_babe	russian_roulette

Sistem berhasil menampilkan pasangan lagu (X, Y) yang mempunyai transisi yang harmonis, dengan syarat yaitu jarak diantara keduanya ≤ 0.4 .

8. Inferensi 8: Rekomendasi Urutan Playlist

▼ Inferensi 8: Rekomendasi Urutan Playlist

Tujuan: Rule 9 – Pasangan lagu ideal untuk urutan playlist

```
rekomenadasi_urutan(X, Y)
```

UJI INFERENSI 8

Ditemukan 1 hasil:

	X	Y
0	good_luck_babe	russian_roulette

Sistem berhasil menampilkan pasangan lagu (X, Y) yang menjadi sebagai rekomendasi urutan, dengan syarat di antara keduanya harus mempunyai transisi yang harmonis

Uji Query Kustom

Query Prolog Kustom

Masukkan Query Prolog (contoh: beda_genre(X,Y).)

```
jarak_graf(X, Y, Jarak).
```

JALANKAN QUERY KUSTOM

Hasil Query Kustom:

	X	Y	Jarak
0	tripping_wires	self_care	0.95
1	tripping_wires	good_luck_babe	0.8
2	good_luck_babe	russian_roulette	0.2
3	beautiful_world	russian_roulette	0.6
4	self_care	good_luck_babe	0.45

Pengguna memasukkan query `jarak_graf(X, Y, Jarak)` untuk melihat bobot tepi graf. Sistem berhasil memproses input teks tersebut, melakukan unifikasi variabel X, Y, dan Jarak, lalu menampilkan seluruh kombinasi yang ada di Knowledge Base.

Bab V: Analisis Kritis dan Penutup

Keterbatasan FOL

Meskipun sistem inferensi yang dibangun berhasil membuktikan validitas aturan transisi lagu, terdapat beberapa keterbatasan mendasar dalam penggunaan *First-Order Logic* (FOL) murni untuk domain musik yang bersifat subjektif dan kontinu:

1. Dalam *Knowledge Base*, atribut energi didefinisikan secara kaku (misalnya: tinggi, sedang, rendah). Pada kenyataannya, sebuah lagu mungkin memiliki energi 75% yang berada di ambang batas. Model logika memaksakan "diskritisasi" ini. Akibatnya, nuansa transisi yang halus seringkali hilang. Predikat `transisi_kasar` mungkin gagal mendeteksi ketidaknyamanan audio yang subtil jika angkanya tidak melampaui *threshold* kaku yang ditetapkan di Rule 6.
2. Aturan `beda_genre(X,Y)` menganggap perpindahan genre sebagai potensi masalah. Namun, bagi pengguna dengan selera eklektik, perpindahan dari Jazz ke Metal mungkin justru diinginkan. Model logika saat ini tidak "belajar" dari riwayat preferensi pengguna, melainkan memaksakan aturan baku "apa yang dianggap transisi yang baik" kepada semua pengguna.
3. Saat ini, fakta (facts) dimasukkan secara manual ke dalam file `prolog_kb.pl`. Untuk aplikasi skala industri dengan jutaan lagu, pendekatan deklaratif murni ini tidak efisien. Diperlukan integrasi dengan *Database Management System* (DBMS) eksternal agar *Logic Programming* hanya menangani aturan, bukan penyimpanan data atribut lagu.
4. Sistem hanya mempertimbangkan tiga dimensi utama: Genre, Energi, dan Tempo. Transisi musik yang mulus juga dipengaruhi oleh kunci nada (*harmonic mixing*), lirik, dan timbre instrumen. FOL sulit memodelkan kemiripan timbre yang kompleks tanpa bantuan pemrosesan sinyal digital (*DSP*).

Rekomendasi Kebijakan

Berdasarkan hasil pengujian inferensi dan analisis akar masalah (*Fishbone Analysis*) pada Bab I, berikut adalah rekomendasi kebijakan teknis untuk pengembangan fitur kurasi playlist di masa depan:

1. Implementasi Fitur "Smart Bridge Insertion". Platform musik harus mengadopsi mekanisme *Intervensi Otomatis* berdasarkan Rule 8 (butuh_lagu_bridge). Sistem harus dapat menawarkan atau menyisipkan "lagu jembatan" di antara keduanya untuk memperhalus gradasi mood jika pengguna menginginkannya.
2. Mengacu pada Rule 9 (rekomendasi_urutan), harus ditambahkan algoritma pengacak berbasis bobot graf (`jarak_graf`). Kebijakan ini memastikan bahwa meskipun urutan lagu tidak terduga, setiap perpindahan lagu tetap berada dalam ambang batas toleransi ($\text{jarak} \leq 0.4$), menjaga koherensi alur playlist.
3. Berdasarkan Rule 3 (wajib_kurasi_manual), platform harus memberikan tanda pada lagu yang terdeteksi sebagai pembentuk_outlier. Ini memberikan edukasi kepada pengguna

bahwa lagu tersebut memiliki potensi merusak alur jika tidak ditempatkan dengan hati-hati.

Kesimpulan

Laporan ini telah berhasil mendemonstrasikan penerapan Logika Informatika dalam menyelesaikan permasalahan nyata pada industri musik digital. Berdasarkan seluruh tahapan implementasi dan pengujian, dapat disimpulkan bahwa:

1. Sistem ini berhasil memitigasi masalah utama yang diidentifikasi dalam diagram Fishbone, yaitu "Tidak adanya alat bantu penyisipan cerdas" dan "Metrik kemiripan yang terlalu sederhana". Dengan memodelkan hubungan antar lagu menggunakan predikat logika, sistem mampu mendeteksi ketidakselarasan yang sering terlewatkan oleh kurasi manual.
2. Formulasi *First-Order Logic* (FOL) terbukti valid dalam merepresentasikan aturan kurasi musik. Pengujian resolusi dan unifikasi menunjukkan bahwa aturan rantai inferensi berjalan konsisten dan bebas dari kontradiksi logika.
3. Integrasi antara Python (Streamlit) dan SWI-Prolog melalui pustaka PySwip berjalan dengan baik. Sistem mampu memberikan umpan balik instan kepada pengguna mengenai kualitas transisi playlist mereka, membuktikan bahwa pendekatan *Logic Programming* dapat diterapkan sebagai mesin backend (*backend engine*) untuk fitur rekomendasi cerdas.
4. Berbeda dengan algoritma *Black Box* (seperti *Deep Learning*), sistem berbasis aturan ini menawarkan transparansi penuh. Pengguna dapat memahami *mengapa* sebuah transisi dianggap buruk (misalnya: "karena beda genre dan energi kontras"), yang merupakan nilai tambah signifikan untuk pengalaman pengguna (*User Experience*).