IF4070 Representasi Pengetahuan dan Penalaran

Ontologi dan Sistem Berbasis Pengetahuan dengan Domain Naruto

Laporan Tugas Proyek 1

Disusun untuk memenuhi tugas mata kuliah Representasi Pengetahuan dan Penalaran pada Semester 1 (satu) Tahun Akademik 2025/2026

Disusun oleh:

Rizqika Mulia Pratama (13522126)
Samy Muhammad Haikal (13522151)
Rafif Ardhinto Ichwantoro (13522159)

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

BANDUNG
2025

DAFTAR ISI

BAB I
DOMAIN

1.1 Deskripsi Domain

Domain yang dipilih adalah dunia Naruto, sebuah universe fiksi dari manga dan anime populer yang diciptakan oleh Masashi Kishimoto. Domain ini berfokus pada berbagai elemen penting yang membentuk dunia tersebut, mulai dari sistem klasifikasi ninja, organisasi-organisasi yang ada, , hingga hubungan antar karakter. Selain itu, domain ini juga mencakup konsep-konsep khas seperti jutsu, klan, tingkatan shinobi, serta peran setiap desa dalam dinamika politik dan militer dunia Naruto.

1.2 Batasan Masalah

Batasan yang diterapkan pada pengerjaan meliputi:

1. **Temporal**: Data diambil dari era Naruto Shippuden hingga awal Boruto
2. **Karakter**: Hanya karakter signifikan yang dimasukkan (±50 individu)
3. **Jutsu**: Fokus pada jutsu ikonik dan kekkei genkai
4. **Simplifikasi**: Tidak mencakup semua detail seperti misi, timeline spesifik, atau transformasi tingkat lanjut
5. **Status**: Hanya 3 status (Alive, Deceased, Sealed).

1.3 Kecocokan Domain

Domain Naruto sangat cocok untuk pengembangan sistem berbasis pengetahuan karena memiliki struktur aturan yang kaku dan hierarkis, yang dapat dimodelkan dengan baik menggunakan logika deskripsi:

1. Hierarki yang Jelas
   Terdapat sistem tingkatan yang tegas (Genin, Chuunin, Jounin, Kage) yang cocok dimodelkan sebagai *subclass*.
2. Logika Elemen (Rule-Based)
   Sistem pertarungan Naruto menggunakan logika batu-gunting-kertas pada elemen chakra yang sangat tepat diimplementasikan menggunakan *rules* Prolog.
3. Pewarisan Sifat (Inheritance)
   Konsep kekkei genkai dan *clan* memungkinkan penerapan *property restrictions* (misal: hanya klan Uchiha yang memiliki mata Sharingan).
4. Klasifikasi Kompleks
   Terdapat banyak parameter untuk mengklasifikasikan entitas, seperti keanggotaan organisasi (Akatsuki), status Jinchuuriki, dan asal desa.

BAB II
ONTOLOGI

2.1 Sumber Pengetahuan

Ontologi ini disusun berdasarkan beberapa sumber utama, yaitu *Naruto Wiki/Fandom* sebagai dokumentasi terperinci mengenai karakter, jutsu, desa, dan lore, serta manga *canon* dari seri *Naruto* (1999–2014) dan *Naruto Shippuden*. Penyusunan ontologi secara khusus tidak memasukkan kejadian, karakter, jutsu, elemen, atau informasi lain yang bersifat *anime-only*, karena konten tersebut tidak dianggap *canon* dalam alur cerita resmi.

2.2 Ontologi

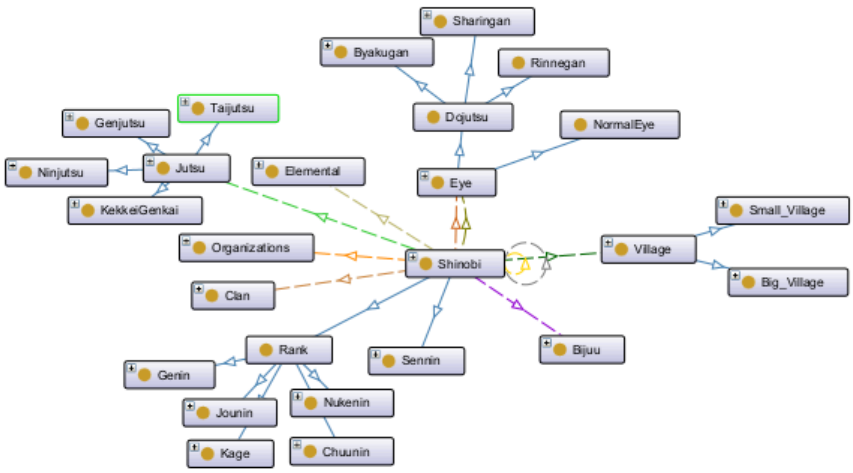
Ontologi terdiri dari 3 elemen yaitu *class*, *object properties*, dan *data properties*. *Class* adalah definisi tipe entitas dalam domain (mis. *Shinobi*, *Village*, *Jutsu*). Kelas mengelompokkan individu yang memiliki karakteristik serupa dan menjadi kerangka struktur untuk representasi pengetahuan.

Object properties adalah hubungan semantik yang menghubungkan dua individu (mis. *isFrom* menghubungkan seorang *Shinobi* dengan sebuah *Village*). Relasi ini memungkinkan penarikan inferensi berbasis koneksi antar-entitas.

Data properties adalah properti yang menghubungkan individu dengan literal/ nilai data (mis. *status* = "Alive" atau *tailNumber* = 8). Digunakan untuk menyimpan informasi primitif yang tidak merupakan entitas tersendiri. Berikut aadalah penjelasan lebih merinci terkait *class/object properties/data properties* yang dibuat.

2.2.1 Class

Kelas-kelas utama yang digunakan untuk merepresentasikan konsep-konsep pada dunia *Naruto*.



Gambar 2.2.1 Class diagram ontologi

- **Shinobi** Entitas utama yang merepresentasikan ninja. Memiliki subclass *rank* untuk memetakan tingkatan ninja pada dunia naruto(contoh: *Genin*, *Chuunin*, *Jonin*, *Kage*, *Sennin*, *Nukenin*). Rank dimodelkan sebagai subclass untuk memudahkan penarikan kesimpulan berdasarkan tipe.

- **Village** Desa-desa pada dunia naruto. dibagi menjadi *Big_Village* dan *Small_Village* sesuai dengan pembagian pada dunia naruto.

- **Elemental** Lima jenis chakra nature (Katon, Suiton, Doton, Futon, Raiton, dan kombinasi elemen seperti Mokuton). Terhubung dengan Shinobi melalui hubungan *hasElement*.

- **Eye** Jenis mata yaitu *Dojutsu* (*Sharingan*, *Byakugan*, *Rinnegan*) dan *NormalEye*. digunakan untuk memetakan jenis-jenis mata yang ada pada dunia naruto.

- **Jutsu** Teknik ninja; memiliki subclass berdasarkan kategori: *Ninjutsu*, *Taijutsu*, *Genjutsu*, serta *Kekkei Genkai* untuk teknik yang diwariskan genetik.

- **Bijuu** Monster berekor (1–9 tails). Entitas ini berhubungan dengan shinobi melalui relasi *isJinchuurikiOf*.

- **Clan** Keluarga ninja (contoh: *Uchiha*, *Hyuuga*) yang dapat memberikan keterkaitan genetik dan restriksi khusus (mis. Semua orang di klan uchiha memiliki sharingan).

- **Organizations** Kelompok/ organisasi khusus (Akatsuki, Seven_sowrdsman_of_the_mist). Terikat dengan kelas *Shinobi* melalui hubungan *IsOrgMemberOf*

2.2.2 Object Properties

Setiap *object property* menghubungkan domain (kelas sumber) dan range (kelas target) untuk menggambarkan hubungan semantik.

- **hasElement: Shinobi → Elemental**. Menyatakan elemen chakra yang dimiliki seorang shinobi. (Contoh: Naruto hasElement Futon)
- **hasEye: Shinobi → Eye**. Menyatakan jenis mata atau dojutsu yang dimiliki seorang shinobi. (Contoh: Sasuke hasEye some Sharingan)
- **hasTeacher / hasStudent: Shinobi ↔ Shinobi** (inverse). Mewakili hubungan guru dan murid. berguna untuk melacak garis pelatihan.
- **masterJutsu: Shinobi → Jutsu**. Menyatakan jutsu yang dikuasai seorang shinobi.
- **isFrom: Shinobi → Village**. Asal desa shinobi.
- **isClanMemberOf: Shinobi → Clan**. Menandakan keangotaan dalam suatu clan.
- **isOrgMemberOf : Shinobi → Organizations**. Menandakan keanggotaan dalam organisasi (mis. Akatsuki).

- **isJinchuurikiOf: Shinobi → Bijuu**. Menandakan shinobi yang menjadi host/ jinchuuriki dari sebuah bijuu.

2.2.3 Data Properties

Data properties menyimpan nilai primitif yang relevan untuk inferensi atau informasi deskriptif.

- **status : string**
  Nilai yang menggambarkan kondisi hidup/keberadaan entitas (*Alive*, *Deceased*, *Sealed*).
- **tailNumber : integer**
  Untuk kelas *Bijuu* menyimpan jumlah ekor (1–9). Berguna untuk validasi (mis. bijuu harus memiliki exactly 1 nilai tailNumber di rentang 1..9).

BAB III
SISTEM BERBASIS PENGETAHUAN

3.1 Arsitektur Sistem

Sistem yang dibuat terdiri dari beberapa bagian antara lain:

1. Knowledge Base: RDF ontology (naruto.rdf)
2. Inference Engine: SWI-Prolog
3. Rule Base: 3 level rules
    i. rules_basic.pl: Basic queries (11 rules)
    ii. rules_combat.pl: Combat & strategy (6 rules)
    iii. rules_inference.pl: Advanced inference (12 rules)
4. User Interface: Command-line interface.

3.2 Fungsionalitas

Level 1: Basic Data Retrieval

Rule	Keterangan
shinobi_info(Name, Village, Rank)	Mengambil profil dasar shinobi berupa desa asal dan rank.
get_clan_members(Clan, Name)	Mengambil daftar anggota dari sebuah clan tertentu.
get_org_members(Org, Name)	Mengambil anggota organisasi (mis. Akatsuki).
get_eye_users(EyeType, Name)	Mendapatkan daftar pengguna dojutsu tertentu.
dual_dojutsu_users(Eye1, Eye2, Name)	Mendapatkan shinobi yang memiliki dua jenis dojutsu sekaligus.
check_status(Status, Name)	Mengecek status hidup/mati/tersegel dari seorang shinobi.
bijuu_info(Jin, Bijuu, TailNum)	Mengambil info jinchuuriki, bijuu yang dikandung, dan jumlah ekor.
who_knows_jutsu(Jutsu, Name)	Menampilkan shinobi yang menguasai jutsu tertentu.
count_elements(Name, Total)	Menghitung jumlah elemen chakra yang dimiliki shinobi.
shinobi_from_small_village(V, S)	Mengambil shinobi yang berasal dari desa kecil.
shinobi_from_big_village(V, S)	Mengambil shinobi yang berasal dari desa besar.

Level 2: Relations & Strategy

Rule	Keterangan
can_defeat(Winner, Loser)	Simulasi hasil pertarungan berdasarkan keunggulan elemen (element wheel).
suggest_counter_element(Enemy, You)	Memberi saran elemen counter terhadap elemen musuh.
find_teacher(Student, Teacher)	Mencari guru langsung dari seorang murid.
find_grand_teacher(Student, Grand)	Mencari "guru dari guru" (grand-teacher).
find_peers(Name, Peer)	Menentukan shinobi yang memiliki level/rank setara.
balanced_team_check(P1,P2,P3,Result)	Mengevaluasi apakah komposisi tim seimbang (elemen, role, atau rank).
element_weakness(Shinobi, WeakElement)	Mencari elemen kelemahan dari seorang shinobi
duo_cover_weakness(S1, S2, ElemS1, ElemS2)	Mencari elemen dari kedua shinobi yang menutupi elemen kelemahan pasanganya.

Level 3: Advanced Inference

Rule	Keterangan
is_legendary(Name)	Menentukan shinobi legendaris (Kage/Sennin dengan minimal 2 elemen).
potential_akatsuki_target(Name, Bij)	Target Akatsuki yang masih hidup, terutama jinchūriki.
high_threat_level(Name, Level)	Analisis tingkat ancaman berdasarkan kekuatan, teknik, afiliasi, atau faktor lain.
kekkei_genkai_user(Name, Jutsu)	Mengidentifikasi pengguna jutsu Kekkei Genkai.
village_protector(Name)	Menentukan Kage aktif yang masih hidup.
village_traitor(Name, Origin)	Daftar nukenin (pengkhianat) beserta desa asal.
akatsuki_candidate(Name)	Kandidat potensial anggota baru Akatsuki.
surviving_akatsuki_member(Name)	Anggota Akatsuki yang masih hidup.
seven_swordsmen_member(Name)	Anggota Seven Swordsmen of the Mist.
surviving_seven_swordsmen(Name)	Anggota Seven Swordsmen yang masih hidup.
rogue_seven_swordsmen(Name)	Anggota Seven Swordsmen yang berstatus nukenin.
loyal_seven_swordsmen(Name)	Anggota Seven Swordsmen yang tetap loyal, bukan nukenin.

3.3 Pemanfaatan Ontologi

Pada Tugas ini fakta dasar tidak ditulis sebagai fakta dasar Prolog biasa, tapi disimpan di file naruto.rdf dalam bentuk RDF/OWL ontology. Prolog tidak menyimpan data manual, tetapi melakukan query ke ontology melalui rdf(Subject, Predicate, Object). Sebagai contoh, pada rule bijuu_info *object properties* dan *data properties* yang di-*import* dari ontologi RDF digunakan untuk mendapatkan bijuu dan Tailnumber dari bijuu tersebut.

```
bijuu_info(Jinchuuriki, BijuuName, TailNumber) :-
    common_prefix(naruto, P),
    atom_concat(P, 'isJinchuurikiOf', IsJin),
    atom_concat(P, 'tailNumber', TailNumProp),

    rdf(S, IsJin, B),
    rdf(B, TailNumProp, literal(type(_, TailAtom))),

    get_name(S, Jinchuuriki),
    get_name(B, BijuuName),

    atom_number(TailAtom, TailNumber).
```

BAB IV
PEMBAHASAN

4.1. Hasil Implementasi Sistem Berbasis Pengetahuan

Sistem berbasis pengetahuan yang dikembangkan menggunakan Prolog berhasil mengintegrasikan ontologi (TBox dan ABox) yang diekspor dari Protege dalam format RDF. Sistem ini mampu menjawab berbagai query mulai dari pencarian fakta sederhana hingga inferensi logika yang kompleks. Berikut adalah pembahasan mengenai fungsionalitas utama sistem:

A. Penelusuran Fakta Dasar (*Basic Retrieval*)

Pada tahap ini, sistem berhasil melakukan ekstraksi data langsung dari graf RDF.

1. Informasi Dasar Shinobi (shinobi_info/3)

```
% Get shinobi info
shinobi_info(Name, Village, Rank) :-
    common_prefix(naruto, P),
    atom_concat(P, 'isFrom', IsFrom),

    rdf(S, IsFrom, D),
    rdf(S, rdf:type, R),

    (   rdf_split_url(_, 'Genin', R)
    ;   rdf_split_url(_, 'Chuunin', R)
    ;   rdf_split_url(_, 'Jounin', R)
    ;   rdf_split_url(_, 'Kage', R)
    ;   rdf_split_url(_, 'Sennin', R)
    ),

    get_name(S, Name),
    get_name(D, Village),
    get_name(R, Rank).
```

Sistem dapat menampilkan profil lengkap seorang shinobi mencakup nama, desa asala, dan *rank*. Aturan ini melakukan *parsing* URI panjang menjadi nama yang mudah dibaca dan memfilter hanya *rank* yang valid (Genin hingga Kage).

- Contoh Query: ?- shinobi_info('Naruto_Uzumaki', Desa, Rank).
- Hasil: Mengembalikan Desa = ‘Konohagakure’ dan Rank = ‘Kage’.

```
?- shinobi_info('Naruto_Uzumaki', Desa, Rank).
Desa = 'Konohagakure',
Rank = 'Kage' ;
Desa = 'Konohagakure',
Rank = 'Sennin' ;
false.
```

2. Identifikasi Anggota Klan dan Organisasi (get_clan_members/2 & get_org_members/2)

```
% Get clan members
get_clan_members(Clan, Member) :-
    common_prefix(naruto, P),
    atom_concat(P, 'isClanMemberOf', IsClan),

    rdf(S, IsClan, K),
    get_name(K, Clan),
    get_name(S, Member).
```

```
% Get Organization members
get_org_members(Org, Member) :-
    common_prefix(naruto, P),
    atom_concat(P, 'isOrgMemberOf', IsOrg),

    rdf(S, IsOrg, O),
    get_name(O, Org),
    get_name(S, Member).
```

Sistem mampu mengelompokkan individu berdasarkan afiliasi mereka. Aturan ini menelusuri properti isClanMemberOf dan isOrgMemberOf untuk menemukan semua anggota dari entitas tertentu.

- Contoh Query: ?- get_clan_members('Uchiha', Nama).
- Hasil: Mengembalikan daftar anggota klan seperti ‘Sasuke_Uchiha’, ‘Itachi_Uchiha’, dll.

```
?- get_org_members('Akatsuki', Member).
Member = 'Deidara' ;
Member = 'Hidan' ;
Member = 'Itachi_Uchiha' ;
Member = 'Kakuzu' ;
Member = 'Kisame' ;
Member = 'Konan' ;
Member = 'Nagato' ;
Member = 'Obito_Uchiha' ;
Member = 'Sasori' ;
Member = 'Yahiko' ;
Member = 'Zetsu'.
```

3. Pencarian Pengguna Dojutsu (get_eye_users/2, dual_dojutsu_users/3)

```
% Get dojutsu users
get_eye_users(EyeType, Shinobi) :-
    common_prefix(naruto, P),
    atom_concat(P, 'hasEye', HasEyeProp),

    rdf(S, rdf:type, TypeNode),

    rdf(TypeNode, owl:onProperty, HasEyeProp),

    (   rdf(TypeNode, owl:someValuesFrom, EyeClassURI)
    ;   rdf(TypeNode, owl:allValuesFrom, EyeClassURI)
    ),

    get_name(EyeClassURI, EyeName),
    get_name(S, Shinobi),

    downcase_atom(EyeName, EyeLower),
    downcase_atom(EyeType, InputLower),
    EyeLower = InputLower.
```

```
dual_dojutsu_users(Eye1, Eye2, ShinobiName) :-
    common_prefix(naruto, P),
    atom_concat(P, 'hasEye', HasEyeProp),

    % First dojutsu restriction
    rdf(S, rdf:type, Restr1),
    rdf(Restr1, owl:onProperty, HasEyeProp),
    (   rdf(Restr1, owl:someValuesFrom, EyeClass1)
    ;   rdf(Restr1, owl:allValuesFrom,  EyeClass1)
    ),

    % Second dojutsu restriction
    rdf(S, rdf:type, Restr2),
    rdf(Restr2, owl:onProperty, HasEyeProp),
    (   rdf(Restr2, owl:someValuesFrom, EyeClass2)
    ;   rdf(Restr2, owl:allValuesFrom,  EyeClass2)
    ),

    % Must be two different dojutsu
    EyeClass1 \= EyeClass2,      You, yesterday •

    % Convert to names
    get_name(EyeClass1, Eye1),
    get_name(EyeClass2, Eye2),
    Eye1 @< Eye2,
    get_name(S, ShinobiName).
```

Implementasi ini menunjukkan kemampuan sistem menangani struktur data kompleks dalam OWL, yaitu owl:Restriction. Sistem menelusuri blank node untuk menemukan individu yang memiliki relasi hasEye dengan kelas mata tertentu (misal: Sharingan atau Rinnegan).

- Contoh Query: ?- get_eye_users('Rinnegan', Nama).
- Hasil: Mengidentifikasi pengguna seperti ‘Nagato’ atau ‘Madara_Uchiha’.

```
?- get_eye_users('Rinnegan', Nama).
Nama = 'Madara_Uchiha' ;
Nama = 'Nagato' ;
Nama = 'Obito_Uchiha' ;
Nama = 'Sasuke_Uchiha' ;
false.
```

- Contoh Query: ?- dual_dojutsu_users(Eye1, Eye2, ShinobiName).
- Hasil: Mengidentifikasi shinobi yang memiliki dua tipe dojutsu berbeda beserta tipe dojutsu yang dimiliki.

```
3 ?- dual_dojutsu_users(Eye1, Eye2, ShinobiName).
Eye1 = 'Rinnegan',
Eye2 = 'Sharingan',
ShinobiName = 'Madara_Uchiha' ;
Eye1 = 'Rinnegan',
Eye2 = 'Sharingan',
ShinobiName = 'Obito_Uchiha' ;
Eye1 = 'Rinnegan',
Eye2 = 'Sharingan',
ShinobiName = 'Sasuke_Uchiha' ;
false.
```

4. Pengecekan Status Kehidupan (check_status/2)

```
% Checking who is still alive or deceased
check_status(Status, Shinobi) :-
    common_prefix(naruto, P),
    atom_concat(P, 'status', StatusProp),

    rdf(S, StatusProp, literal(type(_, Status))),
    get_name(S, Shinobi).
```

Sistem memfilter individu berdasarkan status vitalitas mereka (‘Alive’ atau ‘Deceased’) menggunakan pencocokan literal pada properti status.

- Contoh Query: ?- check_status('Alive', Nama).

```
?- check_status('Alive', Nama).
Nama = 'Choji' ;
Nama = 'Chojuro' ;
Nama = 'Gaara' ;
Nama = 'Hidan' ;
Nama = 'Hinata_Hyuuga' ;
```

5. Informasi Jinchuuriki dan Bijuu (bijuu_info/3)

```
% Finding who is the jinchuuriki and how many the bijuu tails
bijuu_info(Jinchuuriki, BijuuName, TailNumber) :-
    common_prefix(naruto, P),
    atom_concat(P, 'isJinchuurikiOf', IsJin),
    atom_concat(P, 'tailNumber', TailNumProp),

    rdf(S, IsJin, B),
    rdf(B, TailNumProp, literal(type(_, TailAtom))),

    get_name(S, Jinchuuriki),
    get_name(B, BijuuName),

    atom_number(TailAtom, TailNumber).
```

Aturan ini menghubungkan individu dengan Bijuu yang mereka miliki serta mengambil properti data integer (tailNumber) untuk mengetahui jumlah ekor Bijuu tersebut.

- Contoh Query: ?- bijuu_info('Killer_Bee', Bijuu, Ekor).
- Hasil: Bijuu = 'Gyuuki', Ekor = 8.

```
?- bijuu_info('Killer_Bee', Bijuu, Ekor).
Bijuu = 'Gyuuki',
Ekor = 8 ;
false.
```

6. Pencarian Berdasarkan Jutsu (who_knows_jutsu/2)

```
% Finding people who master certain jutsu
who_knows_jutsu(JutsuName, Shinobi) :-
    common_prefix(naruto, P),
    atom_concat(P, 'masterJutsu', MasterProp),
    rdf(S, MasterProp, J),
    get_name(J, JName),
    get_name(S, Shinobi),
    downcase_atom(JName, J1), downcase_atom(JutsuName, J2), J1 = J2.
```

Sistem mencari individu yang menguasai teknik tertentu melalui properti masterJutsu.

- Contoh Query: ?- who_knows_jutsu('Rasengan', Nama).

```
?- who_knows_jutsu('Rasengan', Nama).
Nama = 'Jiraiya' ;
Nama = 'Minato_Namikaze' ;
Nama = 'Naruto_Uzumaki' ;
false.
```

7. Penghitung Penguasaan Elemen (count_elements/2)

```
% Counting how many element a shinobi could mastered
count_elements(Shinobi, Total) :-
    common_prefix(naruto, P),
    atom_concat(P, 'hasElement', HasElem),

    atom_concat(P, Shinobi, S),

    findall(E, rdf(S, HasElem, E), List),
    length(List, Total).
```

Menggunakan fungsi agregasi findall/3 dan length/2 dalam Prolog untuk menghitung total elemen alam (*chakra nature*) yang dikuasai oleh seorang shinobi.

- Contoh Query: ?- count_elements('Kakashi_Hatake', Jumlah).

```
?- count_elements('Kakashi_Hatake', Jumlah).
Jumlah = 1.
```

8. Klasifikasi Berdasarkan Ukuran Desa (shinobi_from_big_village/2 & shinobi_from_small_village/2)

```
% Finding shinobi from big Village
shinobi_from_big_village(Village, Shinobi) :-
    common_prefix(naruto, P),
    atom_concat(P, 'isFrom', IsFrom),
    atom_concat(P, 'Big_Village', Big_Village),
    rdf(S, IsFrom, VillageIRI),

    rdf(VillageIRI, rdf:type, Big_Village),
    get_name(S, Shinobi),
    get_name(VillageIRI, Village).
```

```
% Finding shinobi from small Village
shinobi_from_small_village(Village, Shinobi) :-
    common_prefix(naruto, P),
    atom_concat(P, 'isFrom', IsFrom),
    rdf(S, IsFrom, VillageIRI),
    rdf(VillageIRI, rdf:type, P:'Small_Village'),
    get_name(S, Shinobi),
    get_name(VillageIRI, Village).
```

Sistem membedakan asal shinobi berdasarkan klasifikasi desa (Desa Besar vs Desa Kecil) yang didefinisikan dalam hierarki kelas ontologi.

- Contoh Query: ?- shinobi_from_small_village(Village, Shinobi).

```
4 ?- shinobi_from_small_village(Village, Shinobi).
Village = 'Takigakure',
Shinobi = 'Fuu' ;
Village = 'Yugakure',
Shinobi = 'Hidan' ;
Village = 'Otogakure',
Shinobi = 'Jugo' ;
Village = 'Takigakure',
Shinobi = 'Kakuzu' ;
Village = 'Kusagakure',
Shinobi = 'Karin' ;
Village = 'Amegakure',
Shinobi = 'Nagato' ;
Village = 'Amegakure',
Shinobi = 'Yahiko' ;
false.
```

- Contoh Query: ?- shinobi_from_big_village(Village, Shinobi).

```
5 ?- shinobi_from_big_village(Village, Shinobi)
Village = 'Konohagakure',
Shinobi = 'Choji' ;
Village = 'Kirigakure',
Shinobi = 'Chojuro' ;
Village = 'Iwagakure',
Shinobi = 'Deidara' ;
Village = 'Kirigakure',
Shinobi = 'Fuguki' ;
Village = 'Sunagakure',
Shinobi = 'Gaara' ;
Village = 'Iwagakure',
Shinobi = 'Han' ;
Village = 'Konohagakure',
Shinobi = 'Hashirama' ;
```

B. Relasi dan Pengetahuan Prosedural (*Relational Query*)

Bagian ini mengimplementasikan pengetahuan prosedural yang tidak tertulis secara eksplisit di dalam RDF, melainkan didefinisikan sebagai aturan logika di Prolog.

1. Simulasi Pertarungan (can_defeat/2)

```
% ShinobiX can defeat ShinobiY based on element superiority
can_defeat(ShinobiX, ShinobiY) :-
    common_prefix(naruto, P),
    atom_concat(P, 'hasElement', HasElem),

    rdf(URI_X, HasElem, ElemX),
    rdf(URI_Y, HasElem, ElemY),

    get_name(ElemX, NameX),
    get_name(ElemY, NameY),

    downcase_atom(NameX, E1),
    downcase_atom(NameY, E2),

    superior(E1, E2),

    get_name(URI_X, ShinobiX),
    get_name(URI_Y, ShinobiY).
```

Sistem memprediksi pemenang antara dua shinobi berdasarkan elemen alam yang mereka kuasai. Logika sirkular "Api > Angin > Petir > Tanah > Air > Api" diimplementasikan menggunakan fakta Prolog murni (superior/2) yang dipadukan dengan data elemen dari ontologi.

- Logika: Jika Shinobi A memiliki elemen X dan Shinobi B memiliki elemen Y, dan X superior terhadap Y, maka A menang.

2. Saran Strategi Pertarungan (suggest_counter_element/2)

```
% Element suggestion to counter the enemy's element
suggest_counter_element(EnemyElement, ElementSuggestion) :-
    downcase_atom(EnemyElement, E_Musuh),
    superior(E_Kita, E_Musuh),
    ElementSuggestion = E_Kita.
```

Sistem memberikan rekomendasi taktis. Jika diberikan input elemen musuh, sistem akan menyarankan elemen *counter* yang tepat berdasarkan basis pengetahuan superior/2.

- Contoh Query: ?- suggest_counter_element('Katon', Saran).
- Hasil: Saran = 'suiton'.

```
?- suggest_counter_element('Katon', Saran).
Saran = suiton.
```

3. Penelusuran Hubungan Guru-Murid (find_teacher/2)

```
% Find teacher and student
find_teacher(Student, Teacher) :-
    common_prefix(naruto, P),
    atom_concat(P, 'hasTeacher', HasTeacher),

    rdf(S, HasTeacher, T),
    get_name(S, Student),
    get_name(T, Teacher).
```

Mengambil data relasi langsung antara mentor dan murid menggunakan properti hasTeacher.

4. Penelusuran Silsilah Guru (find_grand_teacher/2)

```
% Find grandteachers of a student
find_grand_teacher(Student, GrandTeacher) :-
    find_teacher(Student, Guru),
    find_teacher(Guru, GrandTeacher).
```

Menunjukkan kemampuan rekursif/berantai sistem. Aturan ini mencari “gurunya guru” dari seorang individu.

5. Pencarian Rekan (find_peers/2)

```
% Shinobi from the same village and the same rank
find_peers(Shinobi, Comrade) :-
    shinobi_info(Shinobi, Desa, Rank),
    shinobi_info(Comrade, Desa, Rank),
    Shinobi \= Comrade.
```

Menemukan shinobi lain yang berasal dari desa yang sama dan memiliki *rank* yang setara.

6. Analisis Keseimbangan Tim (balanced_team_check/4)

```
% A team consist of 3 shinobis is considered as balanced if they have minimum of 3 different element
balanced_team_check(P1, P2, P3, 'BALANCED') :-
    common_prefix(naruto, P), atom_concat(P, 'hasElement', HE),
    findall(E, (rdf(S, HE, E), (get_name(S, P1); get_name(S, P2); get_name(S, P3))), AllElements),
    sort(AllElements, UniqueElements),
    length(UniqueElements, Count),
    Count >= 3.

balanced_team_check(_, _, _, 'IMBALANCE').
```

Sistem mengevaluasi komposisi tim yang terdiri dari tiga orang. Sebuah tim dianggap ‘Balanced’ jika gabungan elemen ketiga anggotanya mencakup minimal 3 jenis elemen yang berbeda, menunjukkan variasi strategi yang baik.

7. Element kelemahan shinobi (element_weakness/2)

```
% Finnding element weakness of a shinobi
element_weakness(Shinobi, WeakElement) :-
    common_prefix(naruto, P),
    atom_concat(P, 'hasElement', HasElem),
    atom_concat(P, Shinobi, S),

    rdf(S, HasElem, ElemURI),
    get_name(ElemURI, ElemName),

    downcase_atom(ElemName, ELower),

    superior(WeakLower, ELower),
    WeakElement = WeakLower.
```

Sistem mencari tahu dan memberikan elemen kelemahan dari shinobi tertentu. Dengan cara membandingkan elemen miliki shinobi, lalu mencari elemen superiornya.

```
5 ?- element_weakness('Naruto_Uzumaki', Weak).
Weak = raiton .
```

8. Dou shinobi yang saling melengkapi(duo_cover_weakness/4)

```
% Finding dual shinobi who cover each other weakness
duo_cover_weakness(S1, S2, ElemS1, ElemS2) :-
    S1 \= S2,
    common_prefix(naruto, P),
    atom_concat(P, 'hasElement', HasElem),

    % Elemen milik S1
    atom_concat(P, S1, URI1),
    rdf(URI1, HasElem, E1URI),
    get_name(E1URI, ElemS1Name),
    downcase_atom(ElemS1Name, E1Lower),

    % Elemen milik S2
    atom_concat(P, S2, URI2),
    rdf(URI2, HasElem, E2URI),
    get_name(E2URI, ElemS2Name),
    downcase_atom(ElemS2Name, E2Lower),

    % S2 punya elemen yang superior terhadap elemen S1
    superior(E2Lower, E1Lower),

    ElemS1 = E1Lower,
    ElemS2 = E2Lower.
```

Sistem memberikan dua elemen dari shinobi dou yang elemen tersebut merupakan elemen kelemahan dari pasangannya.

```
6 ?- duo_cover_weakness('Naruto_Uzumaki','Sasuke_Uchiha',E1,E2).
E1 = doton,
E2 = raiton .
```

C. Inferensi Kompleks (Complex Inference)

Pada tahap ini, sistem menghasilkan pengetahuan baru (*new knowledge*) dengan menggabungkan berbagai fakta dan aturan logika yang rumit.

1. Identifikasi Shinobi Legendaris (is_legendary/1)

```
% Finding a legendary (Shinobi with Kage/Sennin rank and masters at least 2 elements)
is_legendary(Name) :-
    common_prefix(naruto, P),
    atom_concat(P, 'hasElement', HasElem),

    (   rdf(S, rdf:type, Type),
        (get_name(Type, 'Kage') ; get_name(Type, 'Sennin'))
    ),

    findall(E, rdf(S, HasElem, E), ListElemen),
    length(ListElemen, Count),
    Count >= 2,

    get_name(S, Name).
```

Sistem mendefinisikan konsep "Legendaris" bagi shinobi yang memenuhi dua syarat sekaligus: memiliki *rank* tinggi (Kage atau Sennin) dan menguasai minimal 2 elemen alam.

```
?- is_legendary('Jiraiya').
true ;
false.
```

2. Target Potensial Akatsuki (potensial_akatsuki_target/2)

```
% Potential akatsuki target can be defined as a jinchuuriki who is still alive
potential_akatsuki_target(Name, BijuuName) :-
    common_prefix(naruto, P),
    atom_concat(P, 'isJinchuurikiOf', IsJinOf),
    atom_concat(P, 'status', StatusProp),

    rdf(S, IsJinOf, BijuuURI),
    rdf(S, StatusProp, literal(type(_, 'Alive'))),

    get_name(S, Name),
    get_name(BijuuURI, BijuuName).
```

Sistem menyimpulkan target buruan dengan mencari individu yang merupakan Jinchuuriki dan memiliki status *Alive*. Jinchuuriki yang sudah mati (*Deceased*) otomatis diabaikan oleh aturan ini.

```
?- potential_akatsuki_target(Name, BijuuName).
Name = 'Gaara',
BijuuName = 'Shukaku' ;
Name = 'Killer_Bee',
BijuuName = 'Gyuuki' ;
Name = 'Naruto_Uzumaki',
BijuuName = 'Kurama' ;
false.
```

3. Analisis Tingkat Ancaman (high_threat_level/2)

```
% EXTREME: akatsuki member with kekkei genkai
% HIGH: common nukenin
high_threat_level(Name, 'EXTREME') :-
    common_prefix(naruto, P),
    atom_concat(P, 'AkatsukiMember', Akatsuki),
    atom_concat(P, 'KekkeiGenkai', KG), atom_concat(P, 'masterJutsu', MJ),
    rdf(S, rdf:type, Akatsuki), rdf(S, MJ, J), rdf(J, rdf:type, KG),
    get_name(S, Name).
high_threat_level(Name, 'HIGH') :-
    common_prefix(naruto, P), atom_concat(P, 'Nukenin', Nukenin),
    atom_concat(P, 'AkatsukiMember', Akatsuki),
    rdf(S, rdf:type, Nukenin), \+ rdf(S, rdf:type, Akatsuki),
    get_name(S, Name).
```

Sistem mengklasifikasikan tingkat bahaya shinobi ke dalam dua kategori

- EXTREME: Anggota organisasi kriminal (Akatsuki) yang memiliki kemampuan khusus (Kekkei Genkai).
- HIGH: Ninja pelarian (Nukenin) biasa yang bukan anggota Akatsuki.

4. Pencarian Pengguna Kekkei Genkai (kekkei_genkai_user/2)

```
% Find someone with kekkei genkai ability
kekkei_genkai_user(Name, JutsuName) :-
    common_prefix(naruto, P),
    atom_concat(P, 'masterJutsu', MasterProp),
    atom_concat(P, 'KekkeiGenkai', KGClass),

    rdf(S, MasterProp, J),
    rdf(J, rdf:type, KGClass),

    get_name(S, Name),
    get_name(J, JutsuName).
```

Menemukan individu yang menguasai jurus (Jutsu) yang dikategorikan sebagai Kekkei Genkai dalam ontologi.

```
?- kekkei_genkai_user(Name, JutsuName).
Name = 'Itachi_Uchiha',
JutsuName = 'Ametarasu' ;
Name = 'Itachi_Uchiha',
JutsuName = 'Tsukoyomi' ;
Name = 'Nagato',
JutsuName = 'Shinra_Tensei' ;
Name = 'Neji_Hyuuga',
JutsuName = 'Hakkeshō_Kaiten' ;
Name = 'Sasuke_Uchiha',
JutsuName = 'Ametarasu' ;
Name = 'Shisui_Uchiha',
JutsuName = 'Ametarasu' ;
false.
```

5. Identifikasi Pelindung Desa (village_protector/2)

```
% A kage who is still alive
village_protector(Name) :-
    common_prefix(naruto, P),
    atom_concat(P, 'Kage', KageClass),
    atom_concat(P, 'status', StatusProp),

    rdf(S, rdf:type, KageClass),
    rdf(S, StatusProp, literal(type(_, 'Alive'))),

    get_name(S, Name).
```

Menyimpulkan siapa pelindung desa saat ini dengan mencari individu ber-*rank* Kage yang statusnya masih *Alive*

6. Deteksi Pengkhianat Desa (village_traitor/2)

```
% Shinobi from village "X" but with Nukenin rank
village_traitor(Name, AsalDesa) :-
    common_prefix(naruto, P),
    atom_concat(P, 'isFrom', IsFrom),
    atom_concat(P, 'Nukenin', ClassNukenin),

    rdf(S, rdf:type, ClassNukenin),
    rdf(S, IsFrom, D),
    get_name(S, Name),
    get_name(D, AsalDesa).
```

Mengidentifikasi anomali data di mana seseorang terdaftar berasal dari suatu desa (isFrom) namun memiliki tipe kelas Nukenin

7. Kandidat Rekrutmen Akatsuki (akatsuki_candidate/2)

```
% Living nukenin , with S rank (Jounin/Kage/Sennin) but not an akatsuki member yet.
akatsuki_candidate(Name) :-
    common_prefix(naruto, P),
    atom_concat(P, 'AkatsukiMember', ClassAkatsuki),
    atom_concat(P, 'Nukenin', ClassNukenin),

    rdf(S, rdf:type, ClassNukenin),        % Syarat 1: Nukenin
    check_status('Alive', Name),           % Syarat 2: Masih Hidup
    \+ rdf(S, rdf:type, ClassAkatsuki),    % Syarat 3: Belum member Akatsuki

    % Syarat 4: Harus kuat (Jounin/Kage/Sennin)
    (   shinobi_info(Name, _, 'Jounin')
    ;   shinobi_info(Name, _, 'Kage')
    ;   shinobi_info(Name, _, 'Sennin')
    ).
```

Sistem memberikan rekomendasi rekrutmen logis untuk organisasi Akatsuki. Kandidat yang dipilih adalah Nukenin yang masih Alive, memiliki kekuatan

setara *Rank* S (Jounin/Kage/Sennin), namun belum terdaftar sebagai anggota Akatsuki.

8. Member Akatsuki yang masih hidup (surviving_akatsuki_member/1)

```
% Akatsuki member who is still alive
surviving_akatsuki_member(Name) :-
    common_prefix(naruto, P),
    atom_concat(P, 'AkatsukiMember', Akatsuki),
    atom_concat(P, 'status', StatusProp),

    rdf(S, rdf:type, Akatsuki),
    \+ rdf(S, StatusProp, literal(type(_, 'Deceased'))),

    get_name(S, Name).
```

Sistem memberikan member-member akatsuki yang masih hidup. Dimana individu merupakan AkatuskiMember dan status ‘alive’.

9. Member seven swordsmen (seven_swordsmen_member/1)

```
seven_swordsmen_member(Name) :-
    common_prefix(naruto, P),
    atom_concat(P, 'Seven_Swordmans_of_The_Mist', OrgURI),
    get_org_members('Seven_Swordmans_of_The_Mist', Name).
```

Sistem memberikan nama-nama dari shinobi yang merupakan anggota dari organisasi ‘Seven_Swordmans_of_The_Mist’.

```
4 ?- seven_swordsmen_member(Name).
Name = 'Chojuro' ;
Name = 'Fuguki' ;
Name = 'Jinin_Akebino' ;
Name = 'Jinpachi_Munashi' ;
Name = 'Juzo_Biwa' ;
Name = 'Kisame' ;
Name = 'Kushimaru_Kuriarare' ;
Name = 'Mangetsu_Hozuki' ;
Name = 'Raiga_Kurosuki' ;
Name = 'Ringo' ;
Name = 'Zabuza' ;
false.
```

10. Member seven swordsmen yang masih hidup (sruviving_seven_swordsman/1)

```
% Living Seven Swordsmen members
sruviving_seven_swordsmen(Name) :-
    seven_swordsmen_member(Name),
    check_status('Alive', Name).
```

```
2 ?-  surviving_seven_swordsmen(Name).
Name = 'Chojuro' ;
false.
```

Sistem memberikan nama member swordsmen yang masih hidup, dengan mencari individu yang merupakan seven swordsman member dan status ‘Alive’.

11. Seven swordsmen penghianat desa(rogue_seven_swordsmen/1)

```
% Seven Swordsmen who betrayed the Mist (became Nukenin)
rogue_seven_swordsmen(Name) :-
    common_prefix(naruto, P),
    atom_concat(P, 'Nukenin', NukeninClass),

    seven_swordsmen_member(Name),
    atom_concat(P, Name, URI),
    rdf(URI, rdf:type, NukeninClass).
```

Sistem memberikan individu anggota dari seven swordsmen yang menghianati desa Mist dengan mencari individu yang memiliki kelas Nukenin dan merupakan seven swordsmen member.

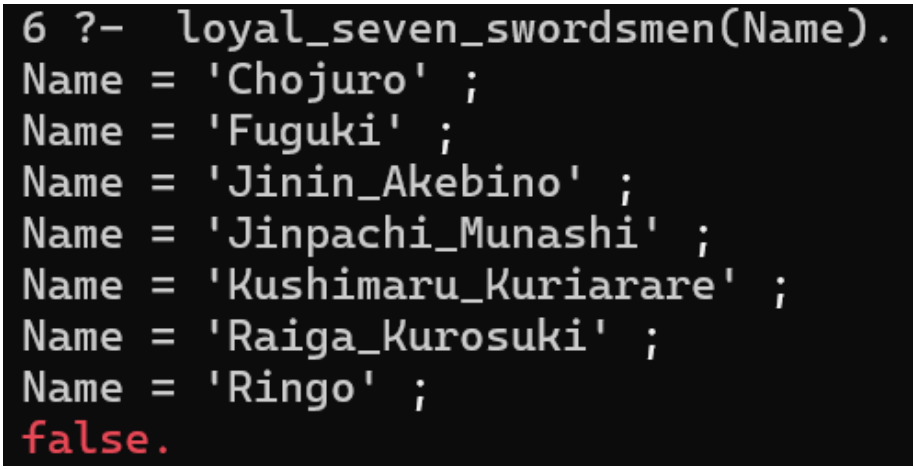
```
7 ?-  rogue_seven_swordsmen(Name).
Name = 'Juzo_Biwa' ;
Name = 'Kisame' ;
Name = 'Zabuza' ;
false.
```

12. Seven swordsmen loyal (loyal_seven_swordsmen/1)

```
% Loyal Seven Swordsmen (not traitors, from Kirigakure)
loyal_seven_swordsmen(Name) :-
    common_prefix(naruto, P),
    atom_concat(P, 'Nukenin', NukeninClass),

    seven_swordsmen_member(Name),
    shinobi_info(Name, 'Kirigakure', _),
    atom_concat(P, Name, URI),
    \+ rdf(URI, rdf:type, NukeninClass).
```

Sistem Mencari shinobi kirigakure yang merupakan member seven swordsmen dan tidak memiliki kelas Nukenin



4.2. Keterlibatan Logika Deskripsi (*Description Logic*)

Pengembangan sistem ini sangat bergantung pada fitur-fitur logika deskripsi yang didefinisikan pada tahap penyusunan ontologi di Protégé:

1. Hirarki Kelas (*Toxonomy*)
   Penggunaan rdfs:subClassOf memungkinkan sistem Prolog untuk melakukan *query* yang fleksibel. Contohnya, ketika mencari Shinobi, sistem secara otomatis menyertakan individu yang berada di subclass Kage, Jounin, atau Genin tanpa perlu menyebutkan satu per satu.
2. *Property Restrictions*
   Salah satu tantangan teknis yang berhasil diatasi adalah penanganan owl:Restriction (seperti someValuesFrom atau allValuesFrom) pada properti hasEye. Karena Protégé menyimpan relasi ini sebagai *blank node* (*struktur* bersarang), aturan Prolog dirancang khusus untuk menelusuri struktur tersebut agar dapat menemukan pengguna mata *Dojutsu* (Sharingan/Rinnegan) dengan tepat.
3. *Domain* dan *Range*
   Definisi domain (misal: Shinobi) dan *Range* (misal: Desa untuk properti isFrom) memastikan konsistensi data saat dilakukan *query*, sehingga entitas yang tidak relevan tidak muncul dalam hasil pencarian.

4.3. Keterbatasan Ontologi dan Sistem

Meskipun sistem berfungsi dengan baik, terdapat beberapa keterbatasan yang ditemukan selama pengembangan:

1. Sifat *Open World Assumption* (OWA)
   Dalam OWL, jika sesuatu tidak dinyatakan “salah”, belum tentu itu “benar”, bisa jadi hanya “tidak diketahui”. Hal ini menyulitkan dalam mendefinisikan aturan negasi yang kuat (misalnya, memastikan seorang Shinobi hanya menguasai satu elemen saja) tanpa menggunakan *closure axiom*.
2. Ketergantungan pada Sintaks RDF

Aturan Prolog sangat terikat pada struktur RDF yang dihasilkan Protégé. Perubahan kecil pada cara Protégé mengekspor data (misalnya perubahan dari RDF ke *Turtle*, atau perubahan penanganan *blank nodes*) dapat menyebabkan aturan Prolog gagal membaca data dan memerlukan penyesuaian kode.

3. Kompleksitas Temporal
   Ontologi yang dibangun bersifat statis. Sistem sulit menangani perubahan status seiring waktu (misalnya, Sasuke yang awalnya Genin Konoha, lalu menjadi Nukenin, dan kemudian membantu aliansi). Saat ini, status ditangani secara sederhana menggunakan properti data status (“Alive”/”Deceased”) tanpa dimensi waktu yang mendalam.

BAB V
KESIMPULAN

Pengembangan sistem berbasis pengetahuan dengan domain Naruto pada tugas proyek ini menunjukkan bagaimana ontologi, logika deskripsi, dan mekanisme inferensi Prolog dapat dikombinasikan untuk membangun sebuah sistem yang mampu melakukan penalaran kompleks secara terstruktur dan konsisten.

Ontologi yang dibangun di Protégé berhasil merepresentasikan dunia Naruto melalui penyusunan kelas, object properties, dan data properties yang terorganisir. Ontologi tersebut tidak hanya berfungsi sebagai kumpulan data statis, tetapi juga menjadi kerangka semantik yang memungkinkan sistem memahami hubungan antar entitas misalnya hubungan shinobi dengan desa, klan, elemen, dojutsu, maupun organisasi. Dengan memanfaatkan struktur RDF/OWL, seluruh pengetahuan dasar dunia Naruto dapat diakses secara fleksibel melalui query Prolog tanpa perlu mendefinisikan ulang fakta secara manual.

Integrasi ontologi dengan SWI-Prolog melalui library semweb menjadikan sistem mampu melakukan penalaran berbasis aturan (rule-based reasoning). Hasil implementasi menunjukkan bahwa sistem mampu menjawab query dari yang paling sederhana hingga inferensi yang memerlukan kombinasi banyak fakta, termasuk struktur OWL seperti someValuesFrom, allValuesFrom, dan hirarki kelas rdfs:subClassOf. Ini membuktikan bahwa ontologi dapat dimanfaatkan sebagai fondasi representasi pengetahuan, sementara Prolog berperan sebagai inference engine yang kuat dalam melakukan reasoning.

Meskipun demikian, beberapa keterbatasan tetap ditemukan, seperti sifat *Open World Assumption* dalam OWL yang menyulitkan penalaran negatif, ketergantungan pada struktur RDF hasil ekspor Protégé, serta tidak adanya dimensi temporal dalam pengetahuan. Namun demikian, keterbatasan tersebut tidak mengurangi keberhasilan utama proyek ini, yaitu menunjukkan bahwa kombinasi ontologi + Prolog dapat menghasilkan sistem penalaran yang konsisten, fleksibel, dan cukup ekspresif untuk domain fiksi yang kompleks seperti dunia Naruto.

BAB VI
DAFTAR PUSTAKA

6.1. Sumber Pustaka

Fandom.com. (2010). *Narutopedia*. [online] Available at: https://naruto.fandom.com/wiki/Narutopedia [Accessed 13 Nov. 2025].

GeeksforGeeks (2018). *Expert Systems in AI*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/artificial-intelligence/expert-systems/ [Accessed 11 Nov. 2025].

Giacomo, G.D. and Maurizio Lenzerini (1996). TBox and ABox Reasoning in Expressive Description Logics. *Proceedings of the 1996 International Workshop on Description Logics, November 2-4, 1996, Cambridge, MA, USA*, [online] 1996, pp.37–48. Available at: https://www.researchgate.net/publication/220957243_TBox_and_ABox_Reasoning_in_Expressive_Description_Logics [Accessed 11 Nov. 2025].

Chan, M. (2025). *What is Ontology and Its Role in Agentic Experience Design?* [online] Salesforce. Available at: https://www.salesforce.com/blog/design-what-is-ontology/ [Accessed 12 Nov. 2025].

6.2. Lampiran

1. Github: https://github.com/rizqikapratamaa/ontology-and-knowledge-based-system
2. Matriks kontribusi

NIM	Tugas
13522126	Merancang Ontologi, Mengembangkan rule Prolog, Laporan bagian Ontologi dan Pembahasan.
13522151	Merancang Ontologi, Mengembangkan rule Prolog, Laporan bagian Domain, Ontologi, dan Sistem berbasis pengetahuan.
13522159	Merancang Ontologi, Mengembangkan rule Prolog, Laporan bagian pembahasan dan kesimpulan.