

DATA SEMANTICS (ECS735P): FINAL

COURSEWORK

The Football Matches Ontology Report



Submitted By:
Name: Aman Naredi
Student Id: 230290873

Table of Contents

1. Core Task (60%)
 - 1.1. Objective
 - 1.2. Ontology Design
 - 1.2.1. Class Hierarchy
 - 1.2.2. Object Properties
 - 1.2.3. Object Characteristics
 - 1.2.4. Data Properties
 - 1.3. Description Logic
 - 1.3.1. DL Axioms and Restrictions
 - 1.4. Ontology Population and Querying
 - 1.5. Justification, Explanation, and Validation of Ontological Modelling Decisions
 - 1.6. Querying to local store
2. Extra Task (20%)
 - 2.1. Objective
 - 2.2. Methodology
 - 2.3. Data Population from WikiData
 - 2.4. Integration of Non-Semantic Datasets
 - 2.5. Querying Local Store
 - 2.6. Conclusion
3. Advance Task (20%)
 - 3.1. Objective
 - 3.2. Implemented SWRL Rules
 - 3.2.1. Rule 1: Highest Scoring Match
 - 3.2.2. Rule 2: Identifying Large Stadium
 - 3.2.3. Rule 3: Indetifying Popular Matches
 - 3.3. Conclusion
4. Source Code and Execution
5. References

1. Core Task (60%)

1.1. Introduction

The purpose of this ontology is to provide a structured representation of information pertaining to football matches. By defining the relationships and attributes crucial for depicting this domain, the ontology aims to facilitate tasks such as match league, match and stadium analysis. It concentrates on fundamental aspects of match-related data, furnishing a robust framework for diverse applications within the sports domain.

The main classes of the ontology include:

Match: This class serves as the central entity within the ontology, representing individual matches that occur within various competitions. Instances of the Match class encapsulate essential information such as match date, attendees, teams involved, and match outcome etc. The Match class forms the core of the ontology, facilitating the organization and analysis of match-related data.

Competition: Instances of the Competition class represent different competitions or leagues within the match domain.

Stadium: The Stadium class represents the venues where matches are held. Instances of this class capture information about stadiums, including name and capacity.

Team: Instances of the Team class represent the individual teams participating in matches.

The football matches ontology uses object properties like **involvesTeam**, **partOfCompetition**, and **isHostedIn** to model class relationships. Through these object properties, the ontology links matches to teams, competitions, and venues, making match-related data organisation and analysis easier.

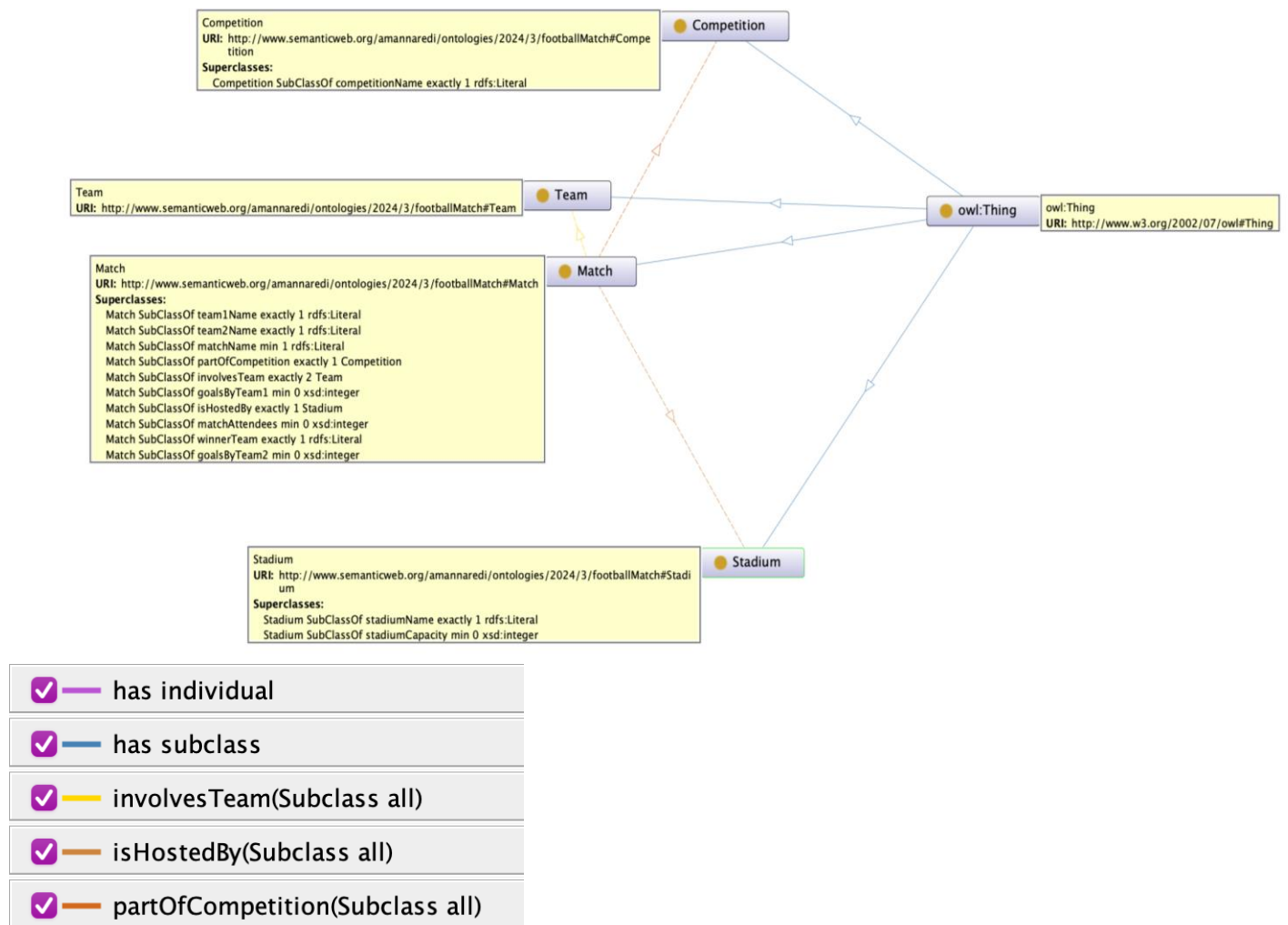
In addition, data properties like **matchDate**, **matchName**, **goalsByTeam1**, **goalsByTeam2**, **matchAttendees**, **matchClass**, **stadiumCapacity**, **stadiumName**, **winnerTeam** etc, capture ontology instance information. These attributes expand the ontology and enable full match data analysis and querying by providing important information such the match date, the number of goals scored by each team, the number of audience, and the winning team.

The football matches ontology's class structure, object and data characteristics, description logic, ontology population, and querying methods will be briefly explained in the following sections. These conversations will show how well the ontology represents football match data.

1.2. Ontology Design

Made use of the Web Ontology Language (OWL2) for the football matches ontology due to its support for Description Logic, and its widespread adoption in the Semantic Web. The T-Box, which defines

the ontology's taxonomy and structure, was created using the Protégé a well-known and often utilised tool for ontology development. The OntoGraph is displayed below.



1.2.1. Class Hierarchy

The ontology consists of four main classes: Match, Competition, Stadium, and Team. The selection of these classes was made to encapsulate fundamental things in the match domain.

1.2.2. Object Properties

The ontology utilises many object properties to construct relationships between instances of distinct classes.

1. **involvesTeam** (Domain: Match, Range: Team):
Represents the relationship between a match and the teams participating.
2. **partOfCompetition** (Domain: Match, Range: Competition):
Represents the relationship between a match and the competition it is associated with.
3. **isHostedIn** (Domain: Match, Range: Stadium):
Represents the relationship between a match and the stadium where it takes place.

These object properties were thoughtfully selected to represent vital relationships within the match domain, with appropriate domains and ranges ensuring accurate connections between instances.

1.2.3. Object Characteristics

The following characteristics were used for the object properties. There are 7 namely: functional, inverse functional, transitive, symmetric, asymmetric, reflexive, and irreflexive. The ones that are used are listed below, along with reasons why other characteristics are not used.

1. involvesTeam (Domain: Match, Range: Team)

- Functional: Not used, as a match typically involves multiple teams.
- Inverse functional: Not used, as a team can be involved in multiple matches.
- Transitive: Not used, because the involvement of teams in matches does not logically extend across matches.
- Symmetric: **Used**, because if a match involves Team A, it necessarily involves another team (e.g., Team B) in the context of that match.
- Asymmetric: Not used, because this property is symmetric.
- Reflexive: Not used, because the relationship is between different entities (matches and teams).
- Irreflexive: **Used**, because a match cannot involve itself as a team.

2. isHostedBy (Domain: Match, Range: Stadium)

- Functional: **Used**, as each match is typically hosted by one stadium.
- Inverse functional: Not used, as a stadium can host multiple matches.
- Transitive: Not used, as hosting does not logically extend across entities.
- Symmetric: Not used, because if a match is hosted by a stadium, the stadium does not similarly 'host' the match.
- Asymmetric: **Used**, since if a match is hosted by a stadium, the reverse cannot be true.
- Reflexive: Not used, because the relationship is between different entities (matches and stadiums).
- Irreflexive: **Used**, because a match cannot host itself.

3. partOfCompetition (Domain: Match, Range: Competition)

- Functional: Not used, as dbpedia was returning two to three names for multiple matches.
- Inverse functional: Not used, as a competition comprises multiple matches.
- Transitive: Not used.
- Symmetric: Not used, because being part of a competition does not imply that the competition is part of the match.

- Asymmetric: **Used**, since if a match is part of a competition, the competition cannot be part of the match.
- Reflexive: Not used, because the relationship is between different entities (matches and competitions).
- Irreflexive: **Used**, because a match cannot be part of itself.

1.2.4. Data Properties

The ontology captures specific attributes of each class through data properties:

1. **competitionName** (Domain: Competition, Range: Literal): Represents the name of the competition.
2. **goalsByTeam1** (Domain: Match, Range: Integer): Represents the number of goals scored by team 1 in a match.
3. **goalsByTeam2** (Domain: Match, Range: Integer): Represents the number of goals scored by team 2 in a match.
4. **matchAttendees** (Domain: Match, Range: Integer): Represents the number of attendees at a match.
5. **matchClass** (Domain: Match, Range: Literal): Represents the classification of a match (e.g., regular season, playoff).
6. **matchDate** (Domain: Match, Range: date): Represents the date on which the match is played.
7. **matchName** (Domain: Match, Range: Literal): Represents the name or title of the match.
8. **stadiumCapacity** (Domain: Stadium, Range: Integer): Represents the maximum seating capacity of the stadium.
9. **stadiumName** (Domain: Stadium, Range: Literal): Represents the name of the stadium where the match is hosted.
10. **team1Name** (Domain: Match, Range: Literal): Represents the name of team 1 participating in the match.
11. **team2Name** (Domain: Match, Range: Literal): Represents the name of team 2 participating in the match.
12. **winnerTeam** (Domain: Match, Range: Literal): Represents the name of the team that won the match.

1.3 Description Logic

We can set axioms, constraints, and limits between classes and properties with DL. This helps us find and fix any possible inconsistencies or wrong representations.

1.3.1. DL Axioms and Restrictions

Implemented the following Description Logic (DL) axioms and restrictions to govern the structure and integrity of the ontology related to football matches:

1. **Every Match must be hosted by exactly one Stadium:**
 - **`isHostedBy exactly 1 Stadium`** (Domain: Match) This restriction ensures that each football match is associated with exactly one stadium, thereby linking each match uniquely to a location where it is held.
2. **Every Match must participate in exactly one Competition:**
 - **`partOfCompetition exactly 1 Competition`** (Domain: Match) This axiom mandates that each match is a part of one, and only one, competition, such as a league or a tournament.
3. **Every Match must have a defined number of attendees:**
 - **`matchAttendees min 0 xsd:integer`** (Domain: Match) This ensures that the number of attendees recorded for a match is at least zero, capturing the case where matches might be played without spectators.
4. **Every Match must have a name:**
 - **`matchName min 1 rdfs:Literal`** (Domain: Match) This ensures that each match has at least a name defined, which is crucial for identification and reference.
5. **Every Match must have two teams:**
 - **`team1Name exactly 1 rdfs:Literal`** (Domain: Match)
 - **`team2Name exactly 1 rdfs:Literal`** (Domain: Match) These constraints ensure that the both the teams are unique.
6. **Every Match must have defined goals for Team 1 and Team 2:**
 - **`goalsByTeam1 min 0 xsd:integer`** (Domain: Match)
 - **`goalsByTeam2 min 0 xsd:integer`** (Domain: Match) These constraints ensure that the goals scored by each team are recorded, starting from zero upwards, which is crucial for match statistics.
7. **Every Match must have a winnerTeam:**
 - **`winnerTeam exactly 1 rdfs:Literal`** (Domain: Match) This ensures that each match has only one winner/draw.
8. **Every Stadium must have a name and a capacity defined:**
 - **`stadiumName exactly 1 rdfs:Literal`** (Domain: Stadium)

- **`stadiumCapacity min 0 xsd:integer`** (Domain: Stadium) These restrictions guarantee that every stadium in the ontology has a unique name and a defined capacity, essential for organizing events and logistical planning.
9. **Every Competition must have a name:**
- **`competitionName min 1 rdfs:Literal`** (Domain: Competition) This ensures that each competition has at least a name defined, which is crucial for identification and reference.

These DL axioms and limits ensure that the ontology is consistent, precise, and detailed enough to handle football match, competition, and related data. This structured approach improves data integrity and facilitates complex football queries and reasoning.

Description: Match

Equivalent To

+

SubClass Of

+

goalsByTeam1	min 0 xsd:integer	?	@	x	o
goalsByTeam2	min 0 xsd:integer	?	@	x	o
isHostedBy	exactly 1 Stadium	?	@	x	o
matchAttendees	min 0 xsd:integer	?	@	x	o
matchName	min 1 rdfs:Literal	?	@	x	o
partOfCompetition	exactly 1 Competition	?	@	x	o
team1Name	exactly 1 rdfs:Literal	?	@	x	o
team2Name	exactly 1 rdfs:Literal	?	@	x	o
winnerTeam	exactly 1 rdfs:Literal	?	@	x	o

Description: Stadium

Equivalent To

+

SubClass Of

+

stadiumCapacity	min 0 xsd:integer	?	@	x	o
stadiumName	exactly 1 rdfs:Literal	?	@	x	o

Description: Competition

Equivalent To

+

SubClass Of

+

competitionName	min 1 rdfs:Literal	?	@	x	o
-----------------	--------------------	---	---	---	---

1.4 Ontology Population and Querying

In the Basic Task, I have populated the Ontology with data from DBPedia.

Source File:

The `basicTask_dbpedia_dataLoad.py` in `.zip` contains the SPARQL query that populates data into our Ontology file (`footballMatch_DB_Populated.owl`)

SPARQL Query:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ex:
<http://www.semanticweb.org/amannaredi/ontologies/2024/3/footballMatch#>

CONSTRUCT {
    ?match rdf:type ex:Match.
    ?match ex:matchName ?matchName.
    ?match ex:matchDate ?matchDate.
    ?match ex:partOfCompetition ?competition.
    ?match ex:isHostedBy ?stadium.
    ?match ex:involvesTeam ?team1.
    ?match ex:involvesTeam ?team2.
    ?match ex:goalsByTeam1 ?goalsByTeam1.
    ?match ex:goalsByTeam2 ?goalsByTeam2.
    ?match ex:matchAttendees ?attendees.
    ?match ex:team1Name ?team1Name.
    ?match ex:team2Name ?team2Name.

    ?team1 rdf:type ex:Team.
    ?team2 rdf:type ex:Team.

    ?stadium rdf:type ex:Stadium.
    ?stadium ex:stadiumName ?stadiumName.
    ?stadium ex:stadiumCapacity ?stadiumCapacity.

    ?competition rdf:type ex:Competition.
    ?competition ex:competitionName ?competitionName.
}
WHERE {
    ?match a dbo:FootballMatch;
        rdfs:label ?matchName;
```

```

        dbo:date ?dateLiteral;
        dbo:team ?team;
        dbp:team1score ?goalsByTeam1;
        dbp:team2score ?goalsByTeam2;
        dbp:attendance ?attendees;
        dbo:location ?stadium;
        foaf:name ?competitionName.
    FILTER (LANG(?matchName) = "en")
    BIND(xsd:dateTime(?dateLiteral) AS ?matchDate)

BIND(URI(CONCAT("http://www.semanticweb.org/amannaredi/ontologies/2024/3/foot
ballMatch#", ENCODE_FOR_URI(?competitionName))) AS ?competition)

    ?team rdfs:label ?teamName.
    FILTER (LANG(?teamName) = "en")

    ?stadium rdfs:label ?stadiumName.

OPTIONAL{
    {
        SELECT ?stadium (MAX(?capacity) AS ?stadiumCapacity)
        WHERE {
            ?stadium dbo:seatingCapacity ?capacity.
        }
        GROUP BY ?stadium
    }
    FILTER (LANG(?stadiumName) = "en")
    {
        SELECT DISTINCT ?match ?team1 ?team2 ?team1Name ?team2Name
        WHERE {
            ?match dbo:team ?team1, ?team2.
            ?team1 rdfs:label ?team1Name.
            ?team2 rdfs:label ?team2Name.
            FILTER (?team1 != ?team2)
            FILTER (?team1Name < ?team2Name) # Ensure team1Name is
lexicographically less than team2Name
            FILTER (LANG(?team1Name) = "en")
            FILTER (LANG(?team2Name) = "en")
        }
    }
}
LIMIT 50

```

This SPARQL query takes data from DBpedia and maps it into my ontology structure set up by the namespace ``http://www.semanticweb.org/amannaredi/ontologies/2024/3/footballMatch#``.

1. CONSTRUCT Clause: It creates new triples (facts) about football matches, teams, stadiums, and events, describing their types and different details such as names, dates, scores, and so on.

2. WHERE Clause: It sorts and links the DBpedia data that is needed: Name, date, scores, participants, and teams that are connected to the match are fetched. Names of matches are checked to make sure they are in English, and date are changed to the right dateTime format.

- Information about the stadium, such as its name and capacity, is retrieved. The capacity is based on the highest number that can be found in the data as it was observed that many stadiums had varying capacities of different years.
- Team details are fetched to make sure that each match has two separate teams, and names are grouped by lexicography to make team1 and team2 stand out.

3. Filters and BINDs: To make sure that all the data is the same, different filters are used. These include language checks and data changes.

4. LIMIT Clause: This limits the query result to 20 matches to keep the size of the output and the time it takes to handle it in check and also for efficient local store querying.

This query is meant to change and combine DBpedia data into a my football matches ontology format. Its main goal is to organise football match data into a structured format that can be queried.

1.5. Justification, Explanation, and Validation of Ontological Modeling Decisions

Class Hierarchy and Properties:

- Structured a comprehensive class hierarchy for the football match ontology, including classes such as Match, Team, Stadium, and Competition, carefully selected relevant object and data properties to effectively capture the attributes of these entities and their relationships, such as **involvesTeam**, **isHostedBy**, and **partOfCompetition**.

Domain and Range Restrictions:

- Domain and range restrictions were carefully applied to ensure correct and logical connections between instances within the ontology classes. This approach helps maintain the integrity of the ontology by ensuring that properties are used consistently and appropriately across different entities.

DL Axioms and Constraints:

- Utilizing Description Logic, defined specific axioms and constraints that govern the relationships between the classes and their properties. These definitions are crucial in

maintaining the consistency of the ontology and accurately representing the knowledge within the football domain.

Ontology Population and Querying:

- Populated the ontology with real-world data extracted from sources like DBpedia by making use of a Python script for data manipulation and SPARQL queries for data integration. This demonstrates the ontology's applicability in real-world scenarios, highlighting how useful it is for answering complicated queries and getting useful information about football matches.

Validation:

- Performed checks and tests to ensure consistency, correctness, and quality, using pellet Reasoner in Protege 5.6.1.

Metrics		Data property axioms	
Axiom	290	SubDataPropertyOf	12
Logical axiom count	229	EquivalentDataProperties	0
Declaration axioms count	61	DisjointDataProperties	0
Class count	4	FunctionalDataProperty	0
Object property count	4	DataPropertyDomain	12
Data property count	13	DataPropertyRange	12
Individual count	41	Individual axioms	
Annotation Property count	1	ClassAssertion	41
Class axioms		ObjectPropertyAssertion	40
SubClassOf	13	DataPropertyAssertion	90
EquivalentClasses	0	NegativeObjectPropertyAssertion	0
DisjointClasses	0	NegativeDataPropertyAssertion	0
GCI count	0	SameIndividual	0
Hidden GCI Count	0	DifferentIndividuals	0
Object property axioms		Annotation axioms	
SubObjectPropertyOf	3	AnnotationAssertion	0
EquivalentObjectProperties	0	AnnotationPropertyDomain	0
InverseObjectProperties	0	AnnotationPropertyRangeOf	0
DisjointObjectProperties	0	<input type="checkbox"/> Synchronising	
FunctionalObjectProperty	0		
InverseFunctionalObjectProperty	0		
TransitiveObjectProperty	0		
SymmetricObjectProperty	0		
AsymmetricObjectProperty	0		
ReflexiveObjectProperty	0		
IrreflexiveObjectProperty	0		
ObjectPropertyDomain	3		
ObjectPropertyRange	3		
SubPropertyChainOf	0		

1.5. Querying local store

The following python script is used to query the local ontology.

```
import rdflib

# Load the RDF file into an RDF graph
rdf_file = "footballMatch_DB_Populated.owl"
graph = rdflib.Graph()
graph.parse(rdf_file, format="xml")

# Define a SPARQL query
query = """
PREFIX ex:
<http://www.semanticweb.org/amannaredi/ontologies/2024/3/footballMatch#>
SELECT DISTINCT ?match ?matchName ?matchDate ?competitionName ?stadiumName
?team1Name ?team2Name ?goalsByTeam1 ?goalsByTeam2 ?attendees ?stadiumCapacity
WHERE {
    ?match rdf:type ex:Match;
           ex:matchDate ?matchDate;
           ex:matchName ?matchName;
           ex:partOfCompetition ?competition;
           ex:isHostedBy ?stadium;
           ex:involvesTeam ?team1;
           ex:involvesTeam ?team2;
           ex:goalsByTeam1 ?goalsByTeam1;
           ex:goalsByTeam2 ?goalsByTeam2;
           ex:matchAttendees ?attendees;
           ex:team1Name ?team1Name;
           ex:team2Name ?team2Name.

    ?team1 rdf:type ex:Team.

    ?team2 rdf:type ex:Team.

    ?stadium rdf:type ex:Stadium;
             ex:stadiumName ?stadiumName;
             ex:stadiumCapacity ?stadiumCapacity.

    ?competition rdf:type ex:Competition;
                 ex:competitionName ?competitionName.
}
"""
```

```
# Execute the query and print the results
results = graph.query(query)
print(f"Total records: {len(results)}")

for row in results:
    print("Match ID:", row["match"])
    print("Match Name:", row["matchName"])
    print("Match Date:", row["matchDate"])
    print("Competition:", row["competitionName"])
    print("Stadium:", row["stadiumName"])
    print("Team 1 Name:", row["team1Name"])
    print("Team 2 Name:", row["team2Name"])
    print("Goals by Team 1:", row["goalsByTeam1"])
    print("Goals by Team 2:", row["goalsByTeam2"])
    print("Attendees:", row["attendees"])
    print("Stadium Capacity:", row["stadiumCapacity"])
    print()
```

Sample Output:

```
Match ID: http://dbpedia.org/resource/1995_Football_League_Trophy_Final
Match Name: 1995 Football League Trophy Final
Match Date: 1995-04-23T00:00:00
Competition: 1995 Football League Trophy Final
Stadium: Wembley Stadium (1923)
Team 1 Name: Birmingham City F.C.
Team 2 Name: Carlisle United F.C.
Goals by Team 1: 1
Goals by Team 2: 0
Attendees: 76663
Stadium Ca Follow link \(cmd + click\)

Match ID: http://dbpedia.org/resource/2010_Football_League_Championship_play-off_Final
Match Name: 2010 Football League Championship play-off Final
Match Date: 2010-05-22T00:00:00
Competition: 2010 Football League Championship play-off Final
Stadium: Wembley Stadium
Team 1 Name: Blackpool F.C.
Team 2 Name: Cardiff City F.C.
Goals by Team 1: 3
Goals by Team 2: 2
Attendees: 82244
Stadium Capacity: 90000

Match ID: http://dbpedia.org/resource/2011_Football_League_Trophy_Final
Match Name: 2011 Football League Trophy Final
Match Date: 2011-04-03T00:00:00
Competition: 2011 Football League Trophy Final
Stadium: Wembley Stadium
Team 1 Name: Brentford F.C.
Team 2 Name: Carlisle United F.C.
Goals by Team 1: 0
Goals by Team 2: 1
Attendees: 40476
Stadium Capacity: 90000
```

Match ID: http://dbpedia.org/resource/1927_FA_Cup_Final
Match Name: 1927 FA Cup Final
Match Date: 1927-04-23T00:00:00
Competition: 1927 FA Cup Final
Stadium: Wembley Stadium (1923)
Team 1 Name: Arsenal F.C.
Team 2 Name: Cardiff City F.C.
Goals by Team 1: 1
Goals by Team 2: 0
Attendees: 91206
Stadium Capacity: 100000

Match ID: http://dbpedia.org/resource/1928_Welsh_Cup_Final
Match Name: 1928 Welsh Cup Final
Match Date: 1928-05-02T00:00:00
Competition: 1928 Football Association of Wales Challenge Cup Final
Stadium: Farrar Road Stadium
Team 1 Name: Bangor City F.C.
Team 2 Name: Cardiff City F.C.
Goals by Team 1: 2
Goals by Team 2: 0
Attendees: 12000
Stadium Capacity: 1500

Match ID: http://dbpedia.org/resource/1977_All-Ireland_Senior_Club_Hurling_Championship_Final
Match Name: 1977 All-Ireland Senior Club Hurling Championship Final
Match Date: 1977-03-27T00:00:00
Competition: 1977 All-Ireland Senior Club Hurling Championship Final
Stadium: Croke Park
Team 1 Name: Glen Rovers GAA
Team 2 Name: Camross GAA
Goals by Team 1: 2
Goals by Team 2: 0
Attendees: 4000
Stadium Capacity: 82300

2. Bonus Task (20%)

2.1. Objective

The objective of this task is to create an ontology that fuses information from at least two distinct external data repositories and answers questions that cannot be answered by either remote knowledge base alone.

2.2. Methodology

In the previous task, I populated my ontology using DBpedia, but DBpedia does not provide details such as the winning team's label and the class of match, specifically whether it is a men's or women's match. To fulfil the requirements of the bonus task, I have utilized Wikidata as one external semantic data source to obtain the winning team's label and match class. Additionally, another data source used is a CSV, which is converted to RDF and then loaded into the ontology.

2.3. Ontology Population from WikiData

Source File:

The `extraTask_wikidataLoad.py` in `.zip` contains the SPARQL query that populates data into our Ontology file (`footballMatch_Populated.owl`).

```
existing_graph = rdflib.Graph()
existing_graph.parse("footballMatch_DB_Populated.owl", format="xml")

# Set up the wikida SPARQL endpoint
sparql = SPARQLWrapper("https://query.wikidata.org/sparql")
sparql.setReturnFormat(RDF)

sparql.setQuery('''
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ex:
<http://www.semanticweb.org/amannaredi/ontologies/2024/3/footballMatch#>
CONSTRUCT {
    ?match rdf:type ex:Match.
    ?match ex:matchName ?matchName.
    ?match ex:matchDate ?matchDate.
    ?match ex:matchClass ?classLabel.
    ?match ex:partOfCompetition ?competition.
    ?match ex:isHostedBy ?stadium.
```



```

    ?match ex:involvesTeam ?team1.
    ?match ex:involvesTeam ?team2.
    ?match ex:matchAttendees ?matchAttendees.
    ?match ex:winnerTeam ?winnerTeamName.

    ?stadium rdf:type ex:Stadium.
    ?stadium ex:stadiumName ?stadiumName.
    ?stadium ex:stadiumCapacity ?stadiumCapacity.

    ?competition rdf:type ex:Competition.
    ?competition ex:competitionName ?competitionName.
}
WHERE {
    ?match wdt:P31 wd:Q17315159; # Instance of a match
        wdt:P2094 ?class;
        wdt:P276 ?stadium;
        wdt:P179 ?competition;
        wdt:P585 ?matchDate;
        wdt:P1346 ?winner;
        wdt:P1110 ?matchAttendees.

    ?stadium wdt:P1083 ?stadiumCapacity;
        wdt:P1705 ?stadiumName.

    SERVICE wikibase:label {
        bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en".
        ?match rdfs:label ?matchName.
        ?class rdfs:label ?classLabel.
        ?competition rdfs:label ?competitionName.
        ?winner rdfs:label ?winnerTeamName.
    }
}
LIMIT 50
'''
)

```

The above SPARQL query was used to get data from the Wikidata source. This data was used to add to the football match ontology with information that wasn't available in DBpedia, like the class of the match (e.g., men's or women's) and the name of the winning team. The `CONSTRUCT` query facilitated the creation of RDF triples directly aligned with our ontology's structure. This fetched graph, containing newly constructed data from Wikidata, was then integrated with our existing RDF graph that was initially populated with DBpedia data. The merged RDF graph, now enriched with comprehensive match details, was subsequently stored in the file named "footballMatch_Populated.owl," effectively enhancing the depth and utility of our ontology.

2.4. Integration of Non-Semantic Datasets

As part of this task, a CSV file containing non-semantic data is generated. The CSV file includes the necessary data to populate the ontology. A Python script utilised for converting CSV files into RDF dumps. The RDF file is subsequently imported into the currently active ontology.

Source File:

matches.csv
non_semanticLoad.py
footballMatch.rdf

```
import csv
from rdflib import Graph, Namespace, URIRef, Literal
from rdflib.namespace import RDF, XSD

fm_ont =
Namespace("http://www.semanticweb.org/amannaredi/ontologies/2024/3/footballMa
tch#")

with open('matches.csv', newline='', encoding='utf-8') as csvfile:
    reader = csv.DictReader(csvfile)
    g = Graph()

    for row in reader:
        match_uri = URIRef(fm_ont[row['matchName']].replace(' ', '_'))
        team1_uri = URIRef(fm_ont[row['team1Name']].replace(' ', '_'))
        team2_uri = URIRef(fm_ont[row['team2Name']].replace(' ', '_'))
        stadium_uri = URIRef(fm_ont[row['stadiumName']].replace(' ', '_'))
        competition_uri = URIRef(fm_ont[row['competitionName']].replace(' ',
'_'))

        g.add((match_uri, RDF.type, fm_ont.Match))
        g.add((team1_uri, RDF.type, fm_ont.Team))
        g.add((team2_uri, RDF.type, fm_ont.Team))
        g.add((stadium_uri, RDF.type, fm_ont.Stadium))
        g.add((competition_uri, RDF.type, fm_ont.Competition))

        g.add((match_uri, fm_ont.matchName, Literal(row['matchName'])))
        g.add((match_uri, fm_ont.matchDate, Literal(row['MatchDate'],
datatype=XSD.dateTime)))
        g.add((match_uri, fm_ont.goalsByTeam1,
Literal(int(row['goalsByTeam1']), datatype=XSD.integer)))
        g.add((match_uri, fm_ont.goalsByTeam2,
Literal(int(row['goalsByTeam2']), datatype=XSD.integer)))
```

```

        g.add((match_uri, fm_ont.matchAttendees,
Literal(int(row['matchAttendees']), datatype=XSD.integer)))
        g.add((match_uri, fm_ont.matchClass, Literal(row['matchClass'])))
        g.add((match_uri, fm_ont.winnerTeam, Literal(row['winnerTeam'])))
        g.add((match_uri, fm_ont.team1Name, Literal(row['team1Name'])))
        g.add((match_uri, fm_ont.team2Name, Literal(row['team2Name'])))
        g.add((stadium_uri, fm_ont.stadiumCapacity,
Literal(int(row['stadiumCapacity']), datatype=XSD.integer)))
        g.add((stadium_uri, fm_ont.stadiumName, Literal(row['stadiumName'])))
        g.add((competition_uri, fm_ont.competitionName,
Literal(row['competitionName'])))

    # Object Properties linking individuals
    g.add((match_uri, fm_ont.involvesTeam, team1_uri))
    g.add((match_uri, fm_ont.involvesTeam, team2_uri))
    g.add((match_uri, fm_ont.isHostedBy, stadium_uri))
    g.add((match_uri, fm_ont.partOfCompetition, competition_uri))

rdf_filename = 'footballMatch.rdf'
g.serialize(destination=rdf_filename, format="turtle") # Turtle is a more
readable format
print(f"RDF data generated and saved to '{rdf_filename}'")

```

This Python script processes data from a CSV file named "matches.csv" and converts this data into RDF format, subsequently storing it in a file called "footballMatch.rdf". The script utilizes the "rdflib" library to construct an RDF graph and populates it with information about football matches and related entities such as teams, stadiums, and competitions. For each record in the CSV, it creates unique URIs for these entities based on their names, and it inserts them into the graph. The script then adds RDF triples to the graph that define the properties of these entities (e.g., match name, date, goals) and the relationships among them (e.g., which team participated in which match, which stadium hosted the match). After all data from the CSV is added to the RDF graph, the graph is serialized and saved in the Turtle format for improved readability.

2.5. Querying Local Store

```
# Define a SPARQL query
PREFIX ex:
<http://www.semanticweb.org/amannaredi/ontologies/2024/3/footballMatch#>
SELECT DISTINCT ?match ?matchDate ?stadiumName ?stadiumCapacity ?winnerTeam
?matchClass
WHERE {
    ?match rdf:type ex:Match;
           ex:matchDate ?matchDate;
           ex:isHostedBy ?stadium;
           ex:winnerTeam ?winnerTeam;
           ex:matchClass ?matchClass.

    ?match2 rdf:type ex:Match;
            ex:matchDate ?matchDate2;
            ex:isHostedBy ?stadium2;
            ex:winnerTeam ?winnerTeam;
            ex:matchClass ?matchClass.

    FILTER(?matchDate = ?matchDate2)

    ?stadium rdf:type ex:Stadium;
             ex:stadiumName ?stadiumName;

    OPTIONAL{
        ?stadium rdf:type ex:Stadium;
                 ex:stadiumCapacity ?stadiumCapacity.
    }
    ?stadium2 rdf:type ex:Stadium;
              ex:stadiumName ?stadiumName2;

    OPTIONAL{
        ?stadium2 rdf:type ex:Stadium;
                  ex:stadiumCapacity ?stadiumCapacity2.
    }

    FILTER(?stadiumName = ?stadiumName2)
}
LIMIT 10
"""
```

```
# Execute the query and print the results
results = graph.query(query)
print(f"Total records: {len(results)}")

for row in results:
    print("Match ID:", row["match"])
    print("Match Date:", row["matchDate"])
    print("Stadium:", row["stadiumName"])
    print("Stadium Capacity:", row["stadiumCapacity"])
    print("Match belongs to class:", row["matchClass"])
    print("Winner Team:", row["winnerTeam"])
    print()
```

Output:

```
Total records: 10
Match ID: http://www.wikidata.org/entity/Q208401
Match Date: 2010-07-11T00:00:00+00:00
Stadium: Soccer City Stadium
Stadium Capacity: 94736
Match belongs to class: men's association football
Winner Team: Spain national association football team

Match ID: http://www.wikidata.org/entity/Q286552
Match Date: 1997-12-21T00:00:00+00:00
Stadium: ملعب الملك فهد الدولي
Stadium Capacity: 92000
Match belongs to class: men's association football
Winner Team: Brazil national football team

Match ID: http://www.wikidata.org/entity/Q303498
Match Date: 1995-01-13T00:00:00+00:00
Stadium: ملعب الملك فهد الدولي
Stadium Capacity: 92000
Match belongs to class: men's association football
Winner Team: Denmark national association football team

Match ID: http://www.wikidata.org/entity/Q389104
Match Date: 1994-07-17T00:00:00+00:00
Stadium: Rose Bowl
Stadium Capacity: 92542
Match belongs to class: men's association football
Winner Team: Brazil national football team

Match ID: http://www.wikidata.org/entity/Q451175
Match Date: 1978-06-21T00:00:00+00:00
Stadium: Estadio Mario Alberto Kempes
Stadium Capacity: 60000
Match belongs to class: men's association football
Winner Team: Austria national association football team

Match ID: http://www.wikidata.org/entity/Q469112
Match Date: 2010-09-03T00:00:00+00:00
Stadium: Råsunda
Stadium Capacity: 36608
Match belongs to class: men's association football
Winner Team: Sweden national association football team

Match ID: http://www.wikidata.org/entity/Q469122
Match Date: 2011-03-29T00:00:00+00:00
Stadium: Johan Cruyff ArenA
Stadium Capacity: 54990
Match belongs to class: men's association football
Winner Team: Netherlands national association football team

Match ID: http://www.wikidata.org/entity/Q509347
Match Date: 2004-07-04T00:00:00+00:00
Stadium: Estádio da Luz
Stadium Capacity: 64642
Match belongs to class: men's association football
Winner Team: Greece national football team

Match ID: http://www.wikidata.org/entity/Q516025
Match Date: 1970-06-17T00:00:00+00:00
Stadium: Estadio Azteca
Stadium Capacity: 115000
Match belongs to class: men's association football
Winner Team: Italy national association football team

Match ID: http://www.wikidata.org/entity/Q525354
Match Date: 1954-06-26T00:00:00+00:00
Stadium: Stade Olympique de la Pontaise
Stadium Capacity: 50000
Match belongs to class: men's association football
Winner Team: Austria national association football team
```

This SPARQL query merges the data from two sources, DBpedia and Wikidata, stored in football matches ontology. It selects matches and stadiums from both sources based on matching dates and stadiums. By ensuring that there's only one unique match for each date and stadium combination, it merges data from both sources effectively.

Performed checks and tests to ensure consistency, correctness, and quality, using pellet Reasoner in Protege 5.6.1. after populating from wikidata and footballMatch.rdf

Ontology metrics:

Metrics

Axiom	1,301
Logical axiom count	1,066
Declaration axioms count	235
Class count	7
Object property count	4
Data property count	13
Individual count	212
Annotation Property count	3

Class axioms

SubClassOf	12
EquivalentClasses	0
DisjointClasses	0
GCI count	0
Hidden GCI Count	0

Object property axioms

SubObjectPropertyOf	3
EquivalentObjectProperties	0
InverseObjectProperties	0
DisjointObjectProperties	0
FunctionalObjectProperty	1
InverseFunctionalObjectProperty	0
TransitiveObjectProperty	0
SymmetricObjectProperty	1
AsymmetricObjectProperty	2
ReflexiveObjectProperty	0
IrreflexiveObjectProperty	2
ObjectPropertyDomain	3
ObjectPropertyRange	3
SubPropertyChainOf	0

Ontology metrics:

Data property axioms

SubDataPropertyOf	12
EquivalentDataProperties	0
DisjointDataProperties	0
FunctionalDataProperty	0
DataPropertyDomain	12
DataPropertyRange	12

Individual axioms

ClassAssertion	212
ObjectPropertyAssertion	217
DataPropertyAssertion	570
NegativeObjectPropertyAssertion	0
NegativeDataPropertyAssertion	0
SameIndividual	0
DifferentIndividuals	0

Annotation axioms

AnnotationAssertion	0
AnnotationPropertyDomain	0
AnnotationPropertyRangeOf	0

☐ Synchronising

2.6. Conclusion

In conclusion, this project successfully demonstrated both the core and extra tasks required for the development and utilization of the ontology within the domain of football matches. The core task involved defining an ontology, including defining the classes, object properties, and their characteristics, along with the implementation of Description Logic (DL) axioms to ensure the logical coherence and integrity of the ontology structure.

Populated the ontology using semantic data sourced from DBpedia, which allowed us to incorporate a semantic dataset covering various aspects of football matches. This population was then verified through the execution of SPARQL queries on the local data store, confirming the functionality and accuracy of the ontology in querying real-world football match data.

The extra task expanded the scope of the ontology by integrating data from other sources. Notably, we utilized Wikidata as an external semantic data source, which provided unique data attributes not available through DBpedia, such as the classification of matches into men's or women's categories and the identification of the winning team. This significantly enables the ontology to answer more complex queries that DBpedia alone could not resolve.

Moreover, we incorporated non-semantic data from a CSV file, which was first converted into RDF format before being directly integrated into the ontology.

This was again verified by querying local store using SPARQL queries and merging the two data sources in the basis of same match date and same stadium considering that on same day in same stadium only one that is a unique match will only be played.

3. Advance Task (20%)

3.1 Objective

To develop a functional ontology using Protégé with SWRL rules and reasoners.

3.2 Implemented SWRL Rules

Made use of **Pellet** as the reasoner for the football matches ontology due to its compatibility with Protégé, native support for SWRL rules, and comprehensive reasoning capabilities.

Note: Use Pellet as the reasoner and run reasoner to view swrl rules results in Protege.

3.2.1. Rule 1: High Scoring Match

Identifying matches in which more than 3 goals were scored.

Explanation:

This rule filters matches in which more than 3 goals were scored by either of the teams, assigns them to a specific category, allowing for easier organisation, querying, and analysis of dominant teams.

SWRL Rule:

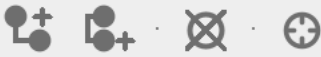

Name
HighScoringMatchTeam1
Comment
Status
Ok
Match(?m) ^ goalsByTeam1(?m, ?score1) ^ swrlb:greaterThan(?score1, 3) -> HighScoringMatch(?m)

Name
HighScoringMatchTeam2
Comment
Status
Ok
Match(?m) ^ goalsByTeam2(?m, ?score1) ^ swrlb:greaterThan(?score1, 3) -> HighScoringMatch(?m)

Output:

After applying the above rule, three such matches were identified. These matches has now been classified under the “HighScoringMatch” category, demonstrating the successful implementation of the rule.

Class hierarchy: HighScoringMatch ? || = ■ ×

 **Asserted** 

- owl:Thing
 - BigStadium
 - Competition**
 - HighScoringMatch**
 - Match**
 - PopularMatch
 - Stadium**
 - Team

Individuals

Individuals (Inferred)

Direct instances (inferred): || = ■ ×

- ◆ **dbpedia:1997_European_Challenge_Cup_Final**
- ◆ **dbpedia:2002_Football_League_Trophy_Final**
- ◆ **dbpedia:Cardiff_Blues_vs_Leicester_Tigers_(2008-09_Heineken_Cup)**
- ◆ **Ligue_1_2023_Paris_Saint-Germain_vs_Lyon**

3.2.2. Rule 2: Finding Large Stadiums

Identifying stadiums which have seating capacity of more than 50,000 and categorizing them as big/large stadiums.

Explanation:

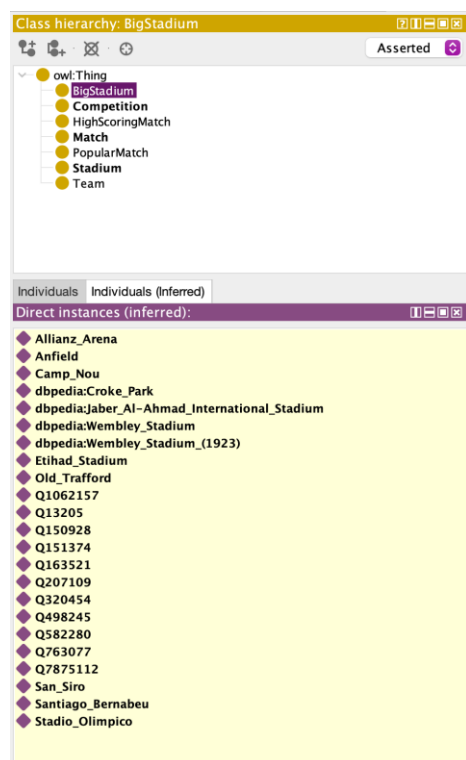
This rule filters stadiums which have seating capacity of more than 50,000, assigns them to a specific category, allowing for easier organisation, querying, and analysis big stadiums.

SWRL Rule:

Name
LargeStadium
Comment
Status
Ok
Stadium(?s) ^ stadiumCapacity(?s, ?c) ^ swrlb:greaterThan(?c, 50000) -> BigStadium(?s)

Output:

After applying the above rule, all such stadiums which have more than 50000 seating capacity were identified. These stadiums has now been classified under the “BigStadium” category, demonstrating the successful implementation of the rule.



3.2.3. Rule 3: Finding Popular Matches

Identifying matches which had more than 50,000 spectators and categorizing them as popular matches.

Explanation:

This rule filters matches which saw more than 50,000 audience, assigns them to a specific category, allowing for easier organisation, querying, and analysis popular matches.

SWRL Rule:

Name
PopularMatch
Comment
Status
Ok
Match(?m) ^ matchAttendees(?s, ?c) ^ swrlb:greaterThan(?c, 50000) -> PopularMatch(?s)

Output:

After applying the above rule, all such matches which had more than 50000 attendees were identified. These matches has now been classified under the “PopularMatches” category, demonstrating the successful implementation of the rule.

The screenshot displays a software interface with two main panels. The left panel, titled 'Class hierarchy: PopularMatch', shows a tree structure of classes. The 'PopularMatch' class is highlighted in purple. The right panel, titled 'Direct instances (inferred):', lists various football matches and their corresponding QIDs. The QID 'Q3298263' is highlighted in purple.

Class hierarchy: PopularMatch

- owl:Thing
 - BigStadium
 - Competition
 - HighScoringMatch
 - Match
 - PopularMatch**
 - Stadium
 - Team

Direct instances (inferred):

- Bundesliga_2023_Bayern_Munich_vs_Borussia_Dortmund
- Champions_League_2023_FC_Barcelona_vs_Real_Madrid_CF
- dbpedia:1922_FA_Cup_Final
- dbpedia:1927_FA_Cup_Final
- dbpedia:1938_FA_Cup_Final
- dbpedia:1995_Football_League_Trophy_Final
- dbpedia:2001_Football_League_First_Division_play-off_Final
- dbpedia:2003_Football_League_Trophy_Final
- dbpedia:2010_Football_League_Championship_play-off_Final
- La_Liga_2023_Real_Madrid_CF_vs_Atletico_Madrid
- Premier_League_2023_Liverpool_FC_vs_Manchester_City_FC
- Premier_League_2023_Manchester_City_FC_vs_Arsenal_FC
- Premier_League_2023_Manchester_United_FC_vs_Chelsea_FC
- Q208401
- Q2299095
- Q2390875
- Q2610314
- Q2696047
- Q2858820
- Q286552
- Q3077917
- Q3225446
- Q3298263**
- Q3298264
- Q389104
- Q469051
- Q469122
- Q509347
- Q516025
- Q585295
- Q586385
- Q610156
- Q63843
- Serie_A_2023_AC_Milan_vs_Juventus_FC
- Serie_A_2023_AS_Roma_vs_SSC_Napoli

3.3. Conclusion

In the specified SWRL rules for the football matches ontology, we effectively employed a combination of classes, data properties, and object properties to derive meaningful inferences:

Rule 1: High Scoring Match

for Team 1

- This rule utilizes the **Match** class along with the **goalsByTeam1** data property to infer high-scoring matches where Team 1 scores more than 3 goals. Instances that meet this criterion are classified under the **HighScoringMatch** class, highlighting matches with significant offensive performances by Team 1.

for Team 2

- Similarly, this rule applies to Team 2 using the **Match** class and the **goalsByTeam2** data property. It identifies matches in which Team 2 scores more than 3 goals, classifying them as **HighScoringMatch**. This allows for an easy query of matches where Team 2 had a strong offensive output.

Rule 2: Large Stadium

- The **LargeStadium** rule uses the **Stadium** class alongside the **stadiumCapacity** data property. It identifies stadiums with a capacity greater than 50,000 as **BigStadium**. This classification is useful for pinpointing major stadiums suitable for hosting large-scale events.

Rule 3: Popular Match

- For matches with a large audience, this rule leverages the **Match** class and the **matchAttendees** data property. Matches with more than 50,000 attendees are classified as **PopularMatch**, indicating a high level of spectator interest and engagement.

Each of these rules leverages different combinations of classes and properties to create specific inferences that enhance the functionality and analytical capabilities of the football matches ontology. Through these rules, we facilitate more nuanced organisation and querying possibilities, such as identifying particularly popular or high-scoring games, thereby enhancing the overall utility of the ontology in sports data analysis.

Active ontology x Entities x Classes x Object properties x Data properties x Individuals by class x DL Query x Individual Hierarchy Tab x SQWRLTab x	
Name	Query
HighScoringMatchTeam1	Match(?m) ^ goalsByTeam1(?m, ?score1) ^ swrlb:greaterThan(?score1, 3) -> HighScoringMatch(?m)
HighScoringMatchTeam2	Match(?m) ^ goalsByTeam2(?m, ?score1) ^ swrlb:greaterThan(?score1, 3) -> HighScoringMatch(?m)
LargeStadium	Stadium(?s) ^ stadiumCapacity(?s, ?c) ^ swrlb:greaterThan(?c, 50000) -> BigStadium(?s)
PopularMatch	Match(?m) ^ matchAttendees(?s, ?c) ^ swrlb:greaterThan(?c, 50000) -> PopularMatch(?s)

4. Source Code and Execution

The source code is stored in the zip file. The .zip contains the following files.

```

|— matches.csv
|— footballMatch.rdf # RDF dump of the local data
|— footballMatch.owl # T-Box
|— footballMatch_DB_Populated.owl # A-Box
|— footballMatch_Populated.owl # A-Box
|— basicTask_dbpedia_dataLoad.py# Python script to populate from DBpedia
|— extraTask_wikiData_Load.py# Python script to populate from Wikidata
|— non_semanticload.py # Python script to populate from non-semantic data
|— basicTaskquery_localStore.py# Querying local ontology
|— extraTaskquery_localStore.py# Querying local ontology after wikidata
|— requirements.txt

```

Commands:

Go to the directory that contains all these files and run the following commands to setup virtualenv.

```

python3 -m venv venv
source venv/bin/activate
# Install requirements
pip install -r requirements.txt

```

To load data from DBpedia into our ontology. Run the following command

```
python basicTask_dbpedia_dataLoad.py
```

To load data from WikiData into our ontology. Run the following command

```
python extraTask_wikiData_Load.py
```

To load data from CSV into our ontology. Run the following command

```
python non_semanticload.py
```

To query local ontology populated by dbpedia. Run the following command

```
python basicTaskquery_localStore.py
```

To query local ontology after populating by wikidata and footballMatch.rdf. Run the following command

```
python extraTaskquery_localStore.py
```

This code takes the csv file and converts it into an RDF file. This file has to be imported into the active ontology under direct imports. The reasoner requires synchronisation for the SWRL inferences to reflect.

5. References

- <https://dbpedia.org/sparql>
- <https://dbpedia.org/ontology/FootballMatch>
- <https://dbpedia.org/ontology/Stadium>
- <https://query.wikidata.org/>
- <https://www.wikidata.org/wiki/Q2736>

End of Paper
