Sivaraman Lakshmipathy
Yatri Modi

**Semantic parsing QA system**
**Report**

**Source code language:** Python 3.7.1
**Parser utilized:** Stanford CoreNLP

**Introduction:**

- The aim of the project is to convert queries in natural language into SQL queries, execute the queries and to return appropriate responses.
- The scope is limited to three databases:
    - Movies: oscar-movie_imdb.sqlite
    - Music: music.sqlite
    - Geography: WorldGeography.sqlite

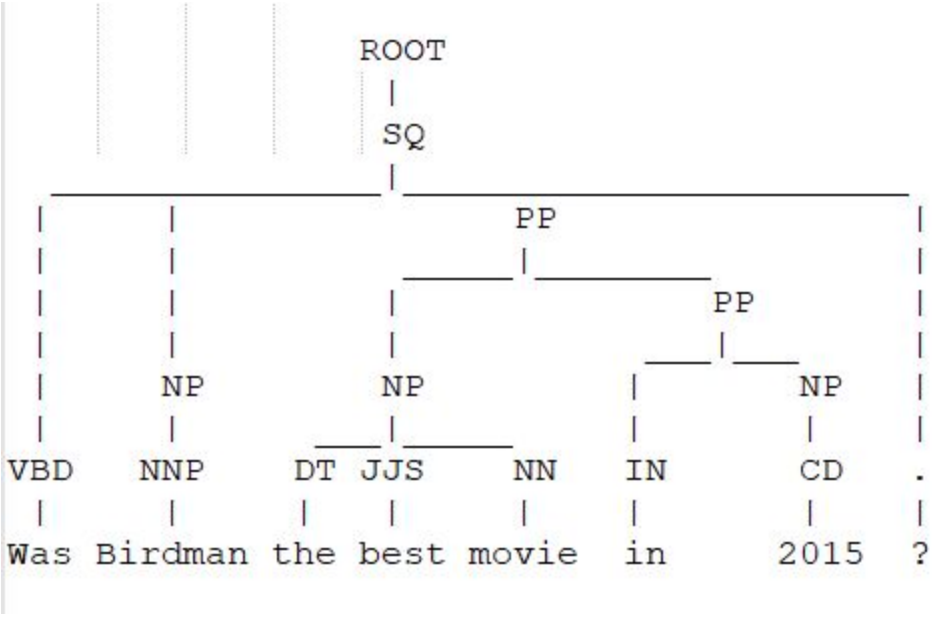
**Design:**

- The design is centered around a depth first traversal of the parse tree generated for each sentence using the Stanford CoreNLP parser.
- The semantic attachments are generated when each transition is encountered in the parse tree. The query object is generated by constructing the sub components for each transition. The components of the query are:
    - SELECT:
        - This component contains the column name to be generated in the case of 'Wh' questions.
        - For 'Yes/No' questions, all the columns are selected using '*' operator.
    - TABLE:
        - Contains the list of tables to be utilized for generating the query.
    - JOIN:
        - Contains the list of joins between the different tables involved in the query.
    - WHERE:
        - Contains all the conditional constraints to be imposed upon the query. There are two types of 'where' clauses:
            - Type 1: The value for the condition matching is provided explicitly.
              Ex: Oscar.Year = 2005
            - Type 2: The value for the condition is to be generated using lambda substitution.
              Ex: Person.Name LIKE '%<LAMBDA_VALUE>%'
    - LAMDBA:
        - Contains all the values to be substituted in the query. They mostly comprise of the Proper Nouns in the statement.
- Once the individual components of the query are generated, they are combined together to form the final query.

Sivaraman Lakshmipathy
Yatri Modi

**Example of query generation:**
Statement: Was Birdman the best movie in 2015?

Parse tree:




| **Transition** | **Semantic Attachment**<br>(in terms of the query components) |
|---|---|
| VBD -> Was | -- |
| NNP -> Birdman | Add ‘Birdman’ to LAMDA (NNP.sem) |
| NP -> NNP | NP.sem = NNP.sem |
| DT -> the | Determiners are ignored. |
| JJS -> best | • Add ‘best’ to ‘WHERE’ component construction.<br>• Add ‘Oscar’ table to ‘TABLE’ component. |
| NN -> movie | Add ‘Movie’ table to ‘TABLE’ component. |
| NP -> DT JJS NN | Add ‘best-movie’ to ‘WHERE’ component construction. |
| IN -> in | -- |

| | |
|---|---|
| CD -> 2015 | • Unsure if the number relates to movie's year or person's year of birth.<br>• On checking the list of tables, there is no person related table. Hence, conclude that the value is related to movies. Add the corresponding 'WHERE' clause. |
| NP -> CD | -- |
| PP -> IN NP | Could be preposition with date/year or place or person. Decide based on child values. Here, it is a date/year value. |
| PP -> NP PP | Join the tables added to the list. |
| . -> ? | Punctuations aren't handled. |
| SQ -> VBD NP PP . | Mark the query type to be 'Yes/No'. This is used to restructure the outputs for the SQL query based on the query type. |

Final components:

```
{
        'LAMBDA': ['birdman'],
        'TABLE': ['OSCAR', 'MOVIE'],
        'WHERE': [      {'column': 'OSCAR.TYPE',
                        'val': 'best-picture',
                        'eql': 'LIKE',
                        'isLambda': False},
                        {'column': 'MOVIE.NAME',
                        'val': '',
                        'eql': 'LIKE',
                        'isLambda': True},
                        {'column': 'OSCAR.YEAR',
                        'val': '2015',
                        'eql': '=',
                        'isLambda': False}],
        'JOIN': [{'t1': 'MOVIE|ID', 't2': 'OSCAR|MOVIE_ID', 'join': 'INNER JOIN', 'op': '='}],
         'SELECT': '*'
}
```

Query type: Yes/No

Sivaraman Lakshmipathy
Yatri Modi

Final query:

```
SELECT *
FROM
        MOVIE T1
        INNER JOIN
        OSCAR T2
        ON T1.ID = T2.MOVIE_ID
WHERE
        T2.TYPE LIKE '%best-picture%'
        AND
        T1.NAME LIKE '%birdman%'
        AND
        T2.YEAR = 2015;
```

Final answer: Yes

**Designing the semantic attachments:**

- The semantic attachments are designed to be compatible with a single pass of the parse tree (DFS) conditional to the query components built till that point.
- For example, when the number ‘2015’ was encountered in the transitions, it was unclear if the value corresponds to the year of birth of a person or the year of release of a movie. Instead of backtracking, a decision can be made based on the query components generated so far.
- The major semantic attachments were linked with the following categories of words:
    - Verbs like ‘win’, ‘direct’: The tables to be queried are identified based on these verbs. For example, the verb ‘win’ includes the ‘Oscar’ table. Similarly, the word ‘direct’ indicates that the ‘Director’ table is to be utilized.
    - Nouns like ‘actor’, ‘movie’: These words are used to identify the values for the ‘WHERE’ clauses to specify the categories.
    - Adjectives like ‘best’, ‘French’: The adjectives are used to add additional ‘WHERE’ clauses. The adjectives related to nationality are converted into ‘WHERE’ components querying the place of birth of the person.

**Challenges:**

- Possessives:
    - Since the tree is parsed in the forward direction only, without backtracking, possessives posed a challenge.
    - For example, “Schindler’s list” is split into Schindler + ‘s + List. The ‘s is encountered after identifying the word ‘Schindler’ as a Proper Noun and adding it to the LAMBDA component. Thus, the latest entry has to be retrieved, updated with the possessive, ‘s, and replaced in the LAMBDA component.

- Embedded ‘S’ in the parse tree:
    - There are parse trees where the ‘S’ node is embedded within other ‘S’ nodes, i.e., sub - statements can be found.
    - This is encountered by parsing the embedded ‘S’ part of the tree separately in addition to the main tree being parsed. The leaf values of the embedded subtree are concatenated together and added to the ‘LAMDBA’ component.
    - Eg:
        - <QUESTION> Does the album Thriller include the track BeatIt?
          <QUERY> SELECT * FROM TRACK T1 INNER JOIN ALBUM T2 ON T1.ALBUMID = T2.ALBUMID WHERE T2.NAME LIKE '%thriller%' AND T1.NAME LIKE '%beatit%';

          <ANSWER> No
        - <QUESTION> Does Thriller include Beat It?

          <QUERY> SELECT * FROM TRACK T1 INNER JOIN ALBUM T2 ON T1.ALBUMID = T2.ALBUMID WHERE T2.NAME LIKE '%thriller%' AND T1.NAME LIKE '%beat it%';

          <ANSWER> Yes
          This query is generated via embedded ‘S’ recursion.
- ‘win’ + category:
    - In order to distinguish between the following queries, ‘WHERE’ components of the semantic attachments had to be modified based on the parse tree.
        - Did Hathaway win an oscar in 2013?
          -> No specific ‘WHERE’ component.
        - Did Hathaway win the oscar for best actress in 2013?
          -> WHERE Oscar.type like ‘%best-actress%
        - Did Hathaway win the oscar for best supporting actress in 2013?
          -> WHERE Oscar.type like ‘%best-supporting-actress%
- Result generation:
    - There are two distinct type of queries: Yes/No and Wh queries. They had to be processed distinctly in order to generate the required results. It was achieved as follows.
    - Yes/No questions were constructed using ‘SELECT *’.
      The final result is ‘Yes’ if there is at least one row in the result set. Else, the final answer is ‘No’.
    - ‘Wh’ questions were constructed using ‘SELECT <TableName.ColumnName>’.
      All the values are retrieved from the result set and concatenated into a string.
      If there are no rows in the result set, the result is “Information not available”.
    - In the case of malformed queries or queries beyond the scope of the domains or in case of exceptions, the result is “I do not know”.

- Domain categorization:
    - In case the categorization of the domain is incorrect, the generated query will be incorrect/malformed and not result in useful results.
    - Additionally, the domain categorization can be tuned to the list of the tables in different domains (databases) so as to generate more accurate results but it hasn't been explored.

**SAMPLE OUTPUT:**

**YES/NO QUESTIONS:**

1. <QUESTION> Was Birdman the best movie in 2015?

   <QUERY> SELECT * FROM MOVIE T1 INNER JOIN OSCAR T2 ON T1.ID = T2.MOVIE_ID WHERE T2.TYPE LIKE '%best-picture%' AND T1.NAME LIKE '%birdman%' AND T2.YEAR = 2015;

   <ANSWER> Yes

2. <QUESTION> Was Beyonce' born in the 1981?

   <QUERY> SELECT * FROM ARTIST T1 WHERE T1.NAME LIKE '%beyonce%' AND T1.DATEOFBIRTH LIKE '%1981%';

   <ANSWER> Yes

3. <QUESTION> Is Rome the capital of Italy?

   <QUERY> SELECT * FROM CAPITALS T1 INNER JOIN COUNTRIES T2 ON T1.COUNTRYID = T2.ID INNER JOIN CITIES T3 ON T3.ID = T1.CITYID WHERE T3.NAME LIKE '%rome%' AND T2.NAME LIKE '%italy%';

   <ANSWER> Yes

Sivaraman Lakshmipathy
Yatri Modi

**WH - QUESTIONS:**

1. <QUESTION> Which actress won the oscar in 2013?

   <QUERY> SELECT T1.NAME FROM PERSON T1 INNER JOIN ACTOR T2 ON T1.ID = T2.ACTOR_ID INNER JOIN OSCAR T3 ON T3.PERSON_ID = T1.ID WHERE T3.TYPE LIKE '%best-actress%' AND T3.YEAR = 2013;

   <ANSWER> Jennifer Lawrence

2. <QUESTION> What is the capital of Italy?

   <QUERY> SELECT T3.NAME FROM CAPITALS T1 INNER JOIN COUNTRIES T2 ON T1.COUNTRYID = T2.ID INNER JOIN CITIES T3 ON T3.ID = T1.CITYID WHERE T2.NAME LIKE '%italy%';

   <ANSWER> Rome

3. <QUESTION> Who sang Thriller?

   <QUERY> SELECT T3.NAME FROM TRACK T1 INNER JOIN ALBUM T2 ON T1.ALBUMID = T2.ALBUMID INNER JOIN ARTIST T3 ON T3.ID = T2.ARTISTID WHERE T1.NAME LIKE '%thriller%';

   <ANSWER> Michael Jackson