Synthesis of SQL Queries from South African Local Language Narrations

George Obaido*, 1, 2, Abejide Ade-Ibijola2, Hima Vadapalli1

1School of Computer Science and Applied Mathematics, University of the Witwatersrand, Johannesburg, 2001, South Africa.
2School of Consumer Intelligence and Information Systems, University of Johannesburg, Johannesburg, 2001, South Africa.

A R T I C L E   I N F O
Article history:
Received: 25 August 2020
Accepted: 18 October 2020
Online: 24 October 2020

Keywords:
Language translation
Synthesis of things
JFA applications
SQL queries

A B S T R A C T
English remains the language of choice for database courses and widely used for instruction in nearly all South African universities, and also in many other countries. Novice programmers of native origins are mostly taught Structured Query Language (SQL) through English as the medium of instruction. Consequently, this creates a myriad of problems in understanding the syntax of SQL as most native learners are not too proficient in English. This could affect a learner’s ability in comprehending SQL syntaxes. To resolve this problem, this work proposes a tool called local language narrations to SQL (Local-Nar-SQL) that uses a type of Finite Machine, such as a Jumping Finite Automaton to translate local language narratives into SQL queries. Further, the generated query extracts information from a sample database and presents an output to the learner. This paper is an extension of work originally presented in a previous study in this field. A survey involving 145 participants concluded that the majority found Local-Nar-SQL to be helpful in understanding SQL queries from local languages. If the proposed tool is used as a learning aid, native learners will find it easier to work with SQL, which will eliminate many of the barriers faced with English proficiencies in programming pedagogies.

1 Introduction

According to Linguists, English has much of its vocabulary borrowed from French and Latin, but it remains a Germanic language in terms of structure and sounds [1, 2]. English is the predominant language of instruction used at most universities and remains the de facto language of communication for many industry sectors [3, 4]. It is interesting to note that many of the world’s population do not speak English as their first language, but it is vastly used for communication and dissemination of knowledge [5, 6]. Most programming and scripting languages, such as C, Java, Python and Structured Query Language (SQL) have keywords, declarations and documentations based in English [6, 7]. Even online resources, such as forums, blogs and developers communities that aid the learning of programming are primarily in English [8, 9]. For example, Stack Overflow, which is one of the largest online communities for developers, enforces that all questions and answers are posted in English. Hence, for a novice programmer to be proficient in programming, English is a necessity [10].

SQL is the most popular query declarative language used for performing operations on relational databases [11, 12]. As a standardised language, many of its syntaxes and statements are English-like, and SQL has found numerous applications in both industry and academia [13, 14]. Listing 1 shows examples of the SQL SELECT query statements that depict English-like formation. These set of queries would retrieve one or more records from a targeted database’s table.

Listing 1: SQL SELECT statement query

```
SELECT * FROM myrecordinfo;
SELECT DISTINCT firstname, lastname
FROM myrecordinfo;
SELECT * FROM myrecordinfo
WHERE firstname='karabo' AND lastname='hlophe';
SELECT * FROM Customers
WHERE country='South Africa'
OR city = 'Pietermaritzburg';
```
South Africa is a multilingual country with eleven official languages [15–16]. Most South African learners prefer to communicate in their local dialects, rather than English [17–16]. Educational instruction for learning programming requires a good proficiency in English, which creates numerous challenges for a local student whose native language is not English [6,18,19]. Just like any other programming language, anecdotal evidence has shown that SQL is hard for students [11–13,20]. Even the straightforward, English-like syntax of SQL is misleading [21,22]. Other studies have identified that the burden of remembering SQL keywords and database schemas are often challenging [20,23]. These challenges have made it crucial for researchers and developers to create culturally-agnostic aids that would assist learners to improve their understanding of SQL. Such systems could be developed to enable learners express their requests in free-forms using local language narrations. To our knowledge, this is the first study that attempts to translate South African local language narrations into SQL queries.

This paper is an extension of work originally presented in [11], where natural language specifications were translated into SQL queries. We extended the use of a Jumping Finite Automaton, a type of Finite Machine for the translation of local language specifications into SQL queries. This idea was further implemented into a desktop-based application called Local-Nar-SQL. The process of translating the local language specifications into SQL queries is presented in Figure 1. Here, a user provides narrations in a local language, that is then processed by Local-Nar-SQL. The system preprocesses and passes the provided narrations to the JFA parser for matches. Further, the tokens are matched and a query is generated, which is made available to the user. In this paper, the following contributions were made. This study has:

1. developed a method for the recognition of local language specifications using a JFA,
2. extended this approach into a tool, called Local-Nar-SQL, for the translation of local language specifications into SQL queries, and
3. evaluated this approach using human subjects and presented initial results.

The rest of this paper is organised as follows: Section 2 highlights the background and related works, Section 3 presents the methodology used for this study, Section 4 shows the implementation and results details, the evaluation information is provided in Section 5 and Section 6 concludes the paper and highlights discussions for future studies.

2 Background

This section presents the problems as research questions, highlights the motivation behind this work and discusses related work.

2.1 Research Questions

This study attempts to solve the following problems:

1. how can we successfully translate local language narrations into SQL queries?
2. can we describe a JFA and implement this into a tool that addresses the above problem?
3. what are human perceptions of the developed tool?

These problems were addressed in Section 3, 4 and 5.

2.2 Motivation

English proficiency is a problem in South Africa English proficiency remains a major concern for local students learning programming [10]. Many programming language constructs, keywords, API methods and libraries are mostly written in English, which pose a great deal of difficulty for a local learner [6,18,19].

Decolonisation of science Issues regarding the decolonisation of the science curriculum is currently being debated in South Africa and the rest of the world [24]. Many local learners agree that the legacies of apartheid (entrenched colonialism) are still evident in education sectors, built by western perspectives, which might impinge access to learning [25,26].

Comprehension problems in SQL Generally, the English-like syntax of SQL is misleading for learners [12,14]. Similarly, most query interfaces are built to report errors in English, which might be confusing for learners to decipher [27,28].

Abstract nature of programming From a novice learners’ perspective, the process of learning a new programming language remains a daunting task [29,30]. Even so, the abstract nature of programming has resulted in high failure rates for Computer Science (CS) disciplines [31,32].
2.3 Why Local Language Narrations

In Section 2.2, it was seen that English language poses numerous difficulties for local language learners. An ideal approach would be to enable such learners to express their intention, free from natural language ambiguities – in narrations. Narrations have been shown to provide textual explanations for programming and scripting language paradigms [11,15,32]. This practice allows learners to express their requests in free-forms, ignoring programming language syntax. Such free-forms allow learners to express their requests in their local language, free from English language ambiguities. For programming languages, especially in low-resource settings, such an approach has been tried in [7,33,44].

2.4 Related Work and Terms

This section presents similar work in the area of language translations and the applications of JFA.

Language Translation An evolutionary method for translating Portuguese language into SQL query [35], a pattern matching approach for translating Hindi language into SQL query [36], Telugu language translation into SQL query [37], translating Italian language to SQL query [38], translating Chinese language into SQL query [39] and Arabic translation into SQL queries [40]–[41].

JFA Applications Abstraction of frequently asked questions (FAQs) for comprehension purposes [42], synthesising SQL queries from narratives [11] and the automatic comprehension of tweets [43].

SQL Comprehension Generating narratives of SQL queries [12], generating narratives of nested SQL queries [14], generating SQL queries from visual specifications [13] and synthesising SQL queries from verbal specifications [44].

The method used in this paper is a JFA, and definitions of this term are as follows.

Definition 1 (A Jumping Finite Automaton (JFA) [45]) A JFA is a five-tuple, \( M = (Q, \Sigma, R, s, F) \) where:

1. \( Q \) is a bounded set of states,
2. \( \Sigma \) is an input alphabet,
3. \( R \) is the bounded set of rules, where \( py \to q \ (p,q \in Q, y\in \Sigma) \),
4. \( s \in Q \) is the start state, and
5. \( F \subseteq Q \) is the final state.

3 Methodology

This section presents the methodology for this study. To begin, local languages were abstracted into a JFA, then queries were normalised, where irrelevant details were removed. Next, a query was generated.

3.1 Abstracting Local Languages to SQL

For local language abstraction into a JFA, entities were identified with colours, such as query types \( (\Sigma_{QT} = a_c) \) in green, attributes \( (\Sigma_{CT} = b_c) \) in red and relations \( (\Sigma_{ET} = e_c) \) in blue. This information is presented in Table 1. The Chinook DB [46] was selected for our use-case with a corpus containing a total of 120 different local languages conversions. This database contains a typical digital music store, consisting of eleven tables and multiple records. Some examples of the JFA abstractions are presented.

<table>
<thead>
<tr>
<th>Table 1: JFA symbols 1 - 30</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_c )</td>
</tr>
<tr>
<td>Khombisa</td>
</tr>
<tr>
<td>Ungitholele</td>
</tr>
<tr>
<td>Ngitholele</td>
</tr>
<tr>
<td>Ukuthola</td>
</tr>
<tr>
<td>Usika</td>
</tr>
<tr>
<td>Fumana</td>
</tr>
<tr>
<td>Khetha</td>
</tr>
<tr>
<td>Enta</td>
</tr>
<tr>
<td>Faka</td>
</tr>
<tr>
<td>Susa</td>
</tr>
<tr>
<td>Funda</td>
</tr>
<tr>
<td>Wys</td>
</tr>
<tr>
<td>Vind</td>
</tr>
<tr>
<td>Lys</td>
</tr>
<tr>
<td>Kry</td>
</tr>
<tr>
<td>Plaas</td>
</tr>
<tr>
<td>Opdateer</td>
</tr>
<tr>
<td>Verwyder</td>
</tr>
<tr>
<td>Skep</td>
</tr>
<tr>
<td>Lees</td>
</tr>
<tr>
<td>Nghenisa</td>
</tr>
<tr>
<td>Sula</td>
</tr>
<tr>
<td>Susa</td>
</tr>
<tr>
<td>Engetela</td>
</tr>
<tr>
<td>Tumbuluxa</td>
</tr>
<tr>
<td>Tkakusisa</td>
</tr>
<tr>
<td>Ulonga</td>
</tr>
<tr>
<td>Ubvisa</td>
</tr>
<tr>
<td>Udadzisa</td>
</tr>
<tr>
<td>Utomola</td>
</tr>
<tr>
<td>Usika</td>
</tr>
</tbody>
</table>

Example 1 above shows a query request sentence in Afrikaans. The English translation for this sentence is: “I want to find all the customer’s details”. The equivalent JFA shows:

\[
M = (\{ I, J, K, L \}, \{ b_{23}, c_{12} \}, R, I, \{ L \})
\]

where \( \{ I, J, K, L \} \) are the states,
\( \{ b_{23}, c_{12} \} \) are the input alphabets,
\( R \) is the set of rules,
\( I \) is a start state, and
\( \{ L \} \) is a final state.

with
The above sentence is the Zulu translation of: “I want to find all the customer’s details”. The equivalent JFA shows:

\[ R = \{ I b_{23} \rightarrow J, J c_{2} \rightarrow K, K a_{12} \rightarrow L \} \]

accepts

\[ L(M) = \{ w \in \{ b_{23}, c_{2}, a_{12} \}^*: |b_{23}| = |c_{2}| = |a_{23}| \} \]
i.e. \( b_{23} = a I; c_{2} = \text{customers}; a_{12} = \text{vind} \)

\[ b_{23}c_{2}a_{12}c_{2}b_{23} \sim b_{23}c_{2}a_{12}Ic_{2} \quad [Ib_{23} \rightarrow J] \]
\[ \sim b_{23}c_{2}a_{12}Kc_{2} \quad [Jc_{2} \rightarrow K] \]
\[ \sim Lb_{23}c_{2}a_{12} \quad [Ka_{12} \rightarrow L] \]

Figure 2 shows an example of a sample JFA with three transitions. The diagram consists of four states, labelled as \( \{ I, J, K, L \} \), where the start state is \( I \) and accepting state is \( L \). \( b_{23} \) begins the movement with the \( \text{al} \) keyword, then \( Jc_{2} \) shows the movement with the \( \text{customer} \) keyword, and the transition, \( Kc_{2} \), ends with the \( \text{vind} \) keyword. Since local languages are rich in vocabulary, the input alphabet may differ in another language, which might result in a change in the states and transitions.

\( \bullet \) Ngifuna ukuthola yonke imininingwane ya ma customer

The above sentence is the Zulu translation of: “I want to find all the customer’s details”. The equivalent JFA shows:

\[ M = \{ \{ I, J, K, L \}, \{ a_{3}, b_{24}, c_{2} \}, R, I; \{ L \} \} \]

where \( \{ I, J, K, L \} \) are the states,
\( \{ a_{3}, b_{24}, c_{2} \} \), are the input alphabets,
\( R \) is the set of rules,
\( I \) is a start state, and
\( L \) is a final state.

with

\[ R = \{ Ia_{3} \rightarrow J, Ib_{24} \rightarrow K, Kc_{2} \rightarrow L \} \]

accepts

\[ L(M) = \{ w \in \{ a_{3}, b_{24}, c_{2} \}^*: |a_{3}| = |b_{24}| = |c_{2}| \} \]
i.e. \( a_{3} = \text{ukuthola}; b_{24} = \text{yonke}; c_{2} = \text{customers} \)

\[ b_{24}a_{3}c_{2}b_{24}c_{2}a_{3} \sim b_{24}a_{3}c_{2}Jb_{24}c_{2} \quad [Ia_{3} \rightarrow J] \]
\[ \sim b_{24}a_{3}c_{2}Kc_{2} \quad [Jb_{24} \rightarrow K] \]
\[ \sim Kc_{2} \rightarrow L \]

Figure 3 shows an example of a JFA with three transitions. The four states as shown in the diagram are \( \{ I, J, K, L \} \), with \( I \) representing the start state and \( L \) denoting the accepting state. \( Ia_{3} \) begins the movement with the \( \text{ukuthola} \) keyword, then \( Ib_{24} \) shows the next input label with the \( \text{yonke} \) keyword, \( Kc_{2} \), ends with the \( \text{customer} \) keyword. It is interesting to note that these examples only cater for a single relation. We may have situations of many states and transitions, taking into account the input statement provided by the learner.

3.2 Normalising Queries

At the normalisation phase, irrelevant details were removed and keywords that appears in our JFA design were used to semantically form a query. The normalisation process is passed to the SQL query generator, which is then used to create and retrieve records from the Chinook DB. As illustrated in Table 2, each local language word is mapped to a query operation. This stage is useful for the SQL generation phase.

<table>
<thead>
<tr>
<th>Keywords</th>
<th>Query Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ukuthola, Thola, Ngtholele,</td>
<td>SELECT</td>
</tr>
<tr>
<td>Fumana, Ngitholela, Vind, Kies</td>
<td></td>
</tr>
<tr>
<td>Al, Alle, Alles, Konke, Yonke,</td>
<td>ALL</td>
</tr>
<tr>
<td>Yothe, Vhothe</td>
<td>CREATE</td>
</tr>
<tr>
<td>Faka, Plaa, Ulonga</td>
<td>INSERT</td>
</tr>
<tr>
<td>Susa, Lees, Verywyder, Utomola,</td>
<td></td>
</tr>
<tr>
<td>Ulvisa Vhothe</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: An example of a sample JFA

www.astesj.com
### 3.3 Generating SQL Queries

This phase takes information from the normalisation phase to semantically generate a query. The generated query is used against the Chinook DB, which then produces an output. Since this task is limited to a single relation, the JFA algorithm was useful for this purpose. Similarly, the task was quite straightforward.

### 4 Implementation and Results

In Section 3, JFA technique was described which showed how a query is generated. Here, the implementation and results of the local language narration process are presented. The JFA approach was implemented using the Microsoft C# .NET framework, which was developed into a software tool, called Local–Nar–SQL.

Local–Nar–SQL was created to allow a native learner to specify a query request in any of the eleven local languages spoken in SA. The software tool uses the learner’s input to generate a query and retrieve records from the sample database. Further, the result is made available to the learner. The sample database used for our use-case was the ChinookDB, which contains multiple records. A sample output of this request is provided in Figure 4.

#### 5 Evaluation

A total of 145 participants participated in a survey that was conducted online. The majority of the participants are native language learners who are familiar with programming and have taken numerous computer science courses. The questions for the survey is available via [https://forms.gle/uSXn8TZpBEWmmGk66](https://forms.gle/uSXn8TZpBEWmmGk66). The results are presented in Figure 5.

Of the responses received, 71% indicated familiarity with SQL, whilst 9% of respondents claimed no familiarity with SQL and 20% were unsure of their answers (Figure 5(a)). The participants were asked about their native language proficiencies. Approximately 29% of learners were proficient in Zulu, 16.6% were proficient in Northern and Southern Sotho, 15.2% claimed to be proficient in Xhosa and 13.1% admitted to being proficient in Afrikaans. The remaining respondents were proficient in Venda (10.3%), Swati (9.7%) or Tsonga (6.9%), and 1.4% of the participants did not list any of these native languages (Figure 5(b)). Furthermore, the participants were asked if the Local–Nar–SQL tool provided a correct translation of local language narrative to SQL: 75.2% affirmed that the tool was accurate, 23.4% were unsure and 1.4% indicated no (Figure 5(c)). When the participants were asked if they think that the tool will assist native learners work with SQL, about 73.8% affirmed that the tool would assist learners, 24.8% were unsure and 1.4% indicated no (Figure 5(d)).

The survey showed that the majority of the learners found the Local–Nar–SQL tool to be useful and helpful towards their interest of learning SQL. If our proposed tool is used as a learning aid, native learners will find it easier to work with SQL, which will eliminate many of the barriers faced with English proficiencies in programming pedagogies.
6 Conclusion and Future Work

This paper presented an extended version of a JFA for the automatic synthesis of SQL queries from local language narrations. This automata-based algorithm was further implemented into a software tool called Local-Nar-SQL, designed for native learners to specify query request in any of the eleven official South African languages of their choice. Local-Nar-SQL takes these requests, generates a query and produces an output that is presented to the learner. If implemented on a large scale, this tool will serve as:

1. a comprehension aid: With Local-Nar-SQL’s textual and translation approaches, it can improve the cognitive workload of a native learner towards understanding the SQL concept.
2. a learning aid: The free-form approach offered by the tool will enable native learners improve and enhance their SQL query skills.
3. a solution to English language barriers: Native learners are not required to attain a high English proficiency to use Local-Nar-SQL. Since local languages are widely spoken in South Africa, it is seen that this tool will stimulate a native learner’s interests in becoming proficient in SQL.

To this end, our evaluation results showed that the majority of learners agreed that this tool will be helpful to native learners yearning to be proficient in SQL. Our evaluation results are consistent with the works of [38] where majority of the native participants agreed that the developed tool helped serve their information needs. In the future, we aim to improve this study to accommodate nested SQL query and other query operation tasks. Also, it would be interesting to conduct a performance evaluation on our proposed tool. Such information would provide us with the accuracy of Local-Nar-SQL.

Conflict of Interest The authors declare no conflict of interest.

Acknowledgment This work is based on research supported by the National Research Foundation (NRF) of South Africa (Grant Number: 119041). Any opinion, findings and conclusions or recommendations expressed in this material are those of the authors and therefore the NRF does not accept liability in regard thereto.

References


