Regular Path Queries in SQLite

Qiwen Wang[1], Hui Lyu[2], Yang Cao[2,3], Bertram Ludäscher[1,2,3]

[1]Department of Computer Science, University of Illinois at Urbana-Champaign, [2]School of Information Sciences, University of Illinois at Urbana-Champaign,
[3] National Center for Supercomputing Applications

Motivation

- **SQLite** is used more than all other database engines combined.
- **Regular Path Queries** (RPQ) make many recursive graph queries easy.

⟹ Goal: Develop a simple **RPQ-to-SQLite compiler**

- Many possible applications, e.g. querying of provenance graphs, workflow graphs, social network graphs, etc.

Challenge

- Should allow **conjunctive RPQs** (CRPQs) to support more queries.
- A variant of RPQs are part of SPARQL and can also be easily implemented in Datalog/Prolog, but widespread deployment of SQLite makes it a more desirable platform.
- Earlier RPQ implementations on top of PostgreSQL were not very efficient.

⟹ Aiming for a **lightweight, reasonably efficient** approach for SQLite.

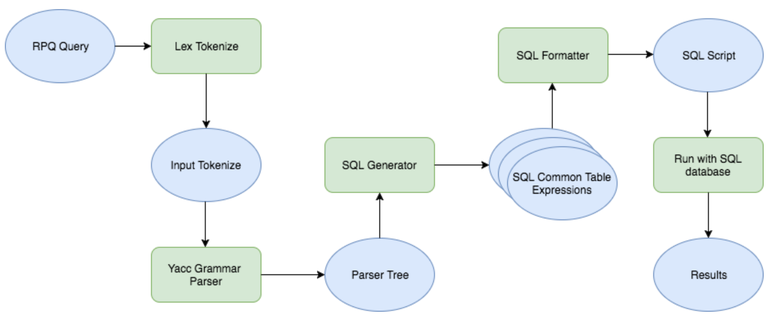

Figure 1. The architecture of the RPQ system.

Implementation

- Tokenize and parse the regular expression query from user input by Lex & Yacc.
- Implement rules and CTE (Common Table Expression) generation for node that starts with "[" and ends with "]", label, concatenation ".", alternation "|", transitive closure " $*$ ", positive transitive closure " $+$ ", inverse " $-$ ", and optional "?".
- Recursively generate strings in Python that represent CTEs recognizable by SQLite, based on rules matched in Yacc.
- Post-order traversal of Yacc parse tree to generate CTEs to resolve dependency issues in SQLite recursion.
- Concatenate all CTEs generated previously and return the result of the last CTE in SQLite.
- Notice that the last CTE corresponds to the root of the parse tree, and thus representing our original query.

Application: Workflows & Provenance

Scientific workflow can often be represented as data graphs that show the relationship between data products and computational steps. Regular path query (RPQ), as a query language for graph-structured data, returns pairs of nodes in a graph that satisfy a regular expression. We will show two examples that query the dependency of the workflow through RPQ.

Example 1: Hamming Numbers

Hamming numbers are positive numbers only with factor 2, 3, or 5. The data graph G_{fish} is the workflow that generates hamming number in increasing order by multiplying either 2, 3, 5. The data graph G_{sail} is the workflow that any number smaller than the current generated hamming number cannot multiply the current number to generate a greater Hamming number.

- **English query**: What are the Hamming numbers along the way to the Hamming number 30?
- **Graphically**: Find all X, such that $[X] - (2|3|5) * -> [30]$
- **RPQ query**: $(2|3|5) * [30]$

The RPQ query returns the starting node and the end node of the path that follow zero or more labels that are either 2, 3, or 5, which is equivalent to our English query. The results of the query are shown in Table 1. The workflow of G_{fish} has more results than the workflow of G_{sail} , because each Hamming number are generated uniquely in G_{sail}, while there are multiple paths to reach a number in G_{fish}.

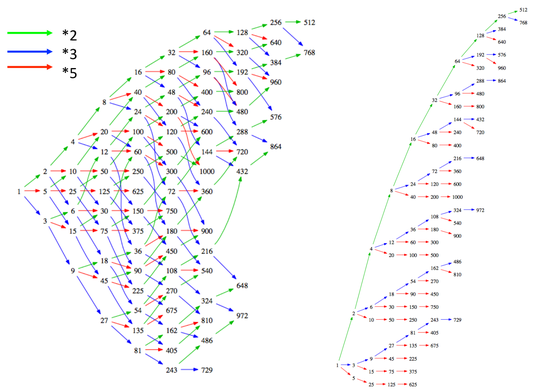

Figure 2. The left side graph is G_{fish}; the right side graph is G_{sail}.

Fish start node	Fish end node	Sail start node	Sail end node
1	30	1	30
2	30	2	30
3	30	6	30
5	30	30	30
6	30		
10	30		
15	30		
30	30		

Table 1. RPQ results after running query $(2|3|5) * [30]$ on G_{fish} and G_{sail}.

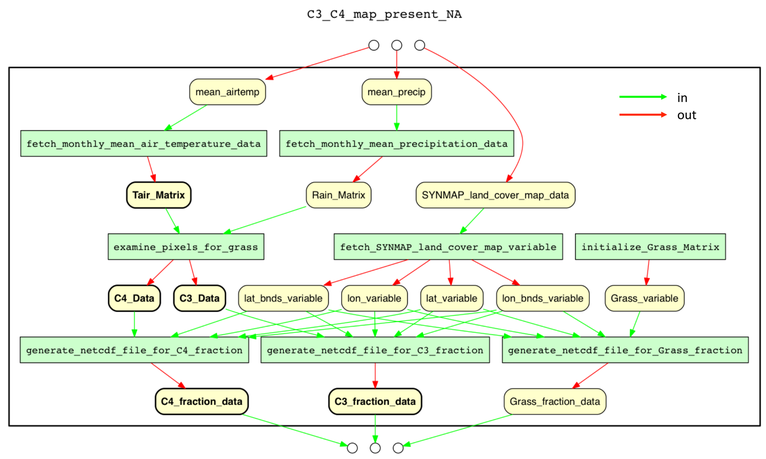

Figure 3. YesWorkflow graph for C3C4.
Remark: Yellow boxes denote data products; green boxes represent computational steps. The bold data products are the RPQ answers to the question: What data products are influenced by **mean_precip**?

YesWorkflow Use Case

YesWorkflow (YW) is a toolkit that enables scientists to annotate their scripts with special comments and automatically generates workflow models (graphs) based on comments. We apply YW in a DataONE use case called C3C4, i.e., a MATLAB script to process climate science datasets (Figure 3).

- **English query**: what are script outputs that are downstream of input data **mean_precip**?
- **RPQ query**: $[\text{mean\_precip}].(in.out)+$

In the workflow graph in C3C4, there is an edge with label "in" from the data product to the step, representing the input of the step, and likewise, there is an edge with label "out" from the step to the data product, representing the output of the step. Thus, $(in.out)$ in the query means the data product directly generated by the involvement of the current data product, and $(in.out)+$ implies all the downstream data of the current data product.

We can see from figure 3 that **mean_precip** has 5 downstream data. Among them, "C3_fraction_data" and "C4_fraction_data" are the outputs of the workflow. From this RPQ, we can see that among three outputs of the workflow, only two of them are related to the input **mean_precip**, that would otherwise hardly be observed only from the original workflow.

Conclusions

In this research, we implemented an RPQ to SQLite compiler that uses CTEs to express recursive queries. This engine can be run on any system that comes with SQLite.

It also supports complex traversals on any type of graph, but is especially useful in answering data dependency queries over workflow and provenance graphs.

Acknowledge

Professor Bertram Ludäscher of School of Information Sciences, University of Illinois at Urbana-Champaign
Funded by NCSA's SPIN (Students Pushing Innovation) program

References

1. Bowers, S., Dey, S.C., Köhler, S., & Ludäscher, B. (2012). Datalog as a Lingua Franca for Provenance Querying and Reasoning. *TaPP*.
2. RPQ to SQLite Engine Project: https://github.com/qwang70/rpq-engine-project
3. Ludäscher B. (2016) A Brief Tour Through Provenance in Scientific Workflows and Databases. In: Lemieux V. (eds) Building Trust in Information.
4. M. L. Y. Wang (2012). Querying Provenance as Regular Path Queries with Relational Databases. Master's thesis, University of California, Davis.
5. Wood, P. T. (2012). Query languages for graph databases. ACM SIGMOD Record, 41(1), 50.
6. Cao, Y., Vu, D., Wang, Q., Zhang, Q., Thavasimani, P., McPhillips, T., Missier, P., Ludäscher, B. (2016). DataONE AHM Provenance Demonstration.