POPL Initial Project Report

dlql (Digital Library Query Language)

Jaffrey Joy
2022201006

Amit Marathe
2022201013

IIIT Hyderabad

1 Objective:

Build an `S-expression` based query language to search the ACM Digital Library with granular filters. Queries can include both **selection** (filter data based on a fixed set of attributes) and **projection** (choose what you want to see about the filtered data). A query result can be reused through **variables** and applying more selection or projection operations on them.

2 Rationale:

The usual approach to search for scientific papers on the ACM Digital Library is to use their search interface. Although it provides an Advanced Search feature, it isn't good enough for performing nested queries with versatile filtering using `AND` and `OR` functionality. It is often the case that one needs to reuse a query by adding some modifications to it with the saved query serving as a basis. For example, a saved search could correspond to "Papers with the anstract containing the term 'lambda calculus' published before 1980". One might want to filter this further based on the author or something else. This is possible in the proposed language using a **selection** operation on the saved query variable.

Generally, it is also the case that one is looking for specific information about a paper like the bibliography of it, the authors, etc. Using the **projection** operation, the user would be able to choose what it wants to see and not a predefined give-everything version which might be overwhelming.

3 Approach:

There would be 2 main modules in the project one in racket and the other in python. We plan to build the **query parser** and **evaluator** in **racket**, as the query language is `S-expression` based. The evaluator would be making appropriate API calls to a **python server** which would then be responsible for sending the corresponding **HTTP queries** and **scraping** the results from the webpage and sending the transformed results back to racket.
An example query would look like the following:

```
(define-query query-name
   (conj (abstract
            (disj "functional"
                  "lambdacalculus"))
          (author "church")))
(run-query (project (list doi) query-name))
```

It can be noticed how a query is being saved to a variable and then being used for a projection operation. Here, `conj` and `disj` refer to `and` and `or` operators respectively.

NOTE: The grammar used for the above query is tentative.

The above query will be parsed and then transformed into its intermediate representation i.e. the URL the API has to hit which is as follows:

```
https://dl.acm.org/action/doSearch?fillQuickSearch=false&target=advanced&expand
    =dl&AllField=Abstract%3A%28%22functional%22+OR+%22lambdacalculus%22%29+AND+
    ContribAuthor%3A%28%22church%22%29
```

While evaluating the ast generated by the query we could also flatten nested queries so that we can reduce the number of HTTP calls to the API endpoint. It could be demonstrated with a trivial example show below:
The following:

```
(conj (conj (abstract
                (disj "functional"
                      "lambdacalculus"))
            (author "church"))
      (conj (abstract "confluence")))
```

will be evaluated such that it was:

```
(conj (abstract
        (disj "functional"
              "lambdacalculus"
              "confluence"))
      (author "church"))
```