COMP2212 Programming Language Concepts Coursework

Semester 2 2017/18

Introduction

In this coursework you are required to *design and implement* a domain specific programming language with the expressive power of *conjunctive queries*. Conjunctive queries—introduced in more detail below— are first-order logic formulas that use only relation symbols, equality, conjunction, and existential quantification. They are a kernel language of queries to relational databases and act as the foundations of several languages: they are select-project-join queries in *relational algebra*, select-from-where queries in *SQL*, as well as representing an important fragment of the popular language *datalog*. You are welcome to research any of the above to inspire the design of your own language. If you want, you could stick closely to the syntax of the logic of conjunctive queries. Of course, you are also more than welcome to go your own way and be as original and creative as you want: it is *YOUR* programming language, and your design decisions.

You are required to (1) invent an appropriate syntax and (2) write an interpreter (possibly using `Alex` and `Happy` for lexing and parsing). Your overall goal is to be

able to write a program for any particular conjunctive query.

For the five example problems listed below, you will be required to produce a program, in your language, that solves each of the problems. These five programs, together with the Haskell sources for your interpreter, are the required deliverables for the first submission.

Please keep a record of all the sources you consult that influence your design, and include them in your programming language manual. The manual will be required for the second submission, along with programs for five additional unseen problems, which we will make public *after* the deadline for the first submission. You will find more procedural details at the end of this document.

The specification is deliberately loose. If we haven't specified something, it is a design decision for you to make: e.g. how to handle syntax errors, illegal inputs, whether to allow comments, support for syntax highlighting, compile time warnings, any type systems etc. A significant part (30%) of the mark will be awarded for these qualitative aspects, where we will also take into account the design of the syntax and reward particularly creative and clean solutions with additional marks. The remaining 70% of the mark will be awarded for correctly solving a number of problems using your programming language. The correctness of your solution will be checked with a number of automated tests. We will release an "automarker" script before the first deadline which will give you an indication as to how your programs will perform in our test harness.

Conjunctive queries

Assume a set Σ of relation symbols with arity function $ar : \Sigma \to \mathbb{N}$ and a countable set $Var = \{x_i \mid i \in \mathbb{N}\}$ of variables. The recursive grammar for the calculus of conjunctive queries is:

$$\Phi ::= \Phi \wedge \Phi \mid x_i = x_j \mid R(\vec{x}) \mid \exists x.\Phi \tag{1}$$

where $R \in \Sigma$, $ar(R) = n$, and $\vec{x}$ is a list of length n of variables from Var. A variable that appears under the scope of an existential quantifier is said to be *bound*, otherwise it is *free*.

For the coursework, we will use formula judgements. A judgement, in general, will look like

$$\vec{x} \vdash \varphi$$

where φ is a formula obtained from the grammar above and $\vec{x}$ is a list of *all* the free variables used in φ. For the purposes of this coursework, we assume that every free variable appears in the scope of at least one relation symbol: so, for, example, we allow the judgement

$$x_1, x_2 \vdash P(x_1) \wedge Q(x_2) \wedge x_1 = x_2$$

where P and Q are assumed to be unary relation symbols, since x_1 appears in the scope of P and x_2 in the scope of Q. However, we do *not* allow the judgement

$$x_1, x_2 \vdash P(x_1) \wedge x_1 = x_2$$

since here x_2 does not appear in the scope of a relation symbol.

Problems

For every relation symbol R used in a formula, you may assume that we will place a CSV file called `R.csv` in the same directory where we execute your interpreter. The file will always be compatible with the arity of R, so, e.g., if R is a relation symbol of arity three then every row of `R.csv` will contain three comma-separated values. You can treat each value as a string and—since our specification does not go beyond conjunctive queries—assume that we will not require you to perform any additional operations on values (e.g. you do not need to worry about being asked to perform string concatenation, arithmetic operations in case values are integers, etc.). The semantics of formulas ought to be clear from the explanations in the five problems below: in case of any confusion, please email Pawel asap.

Problem 1 - Conjunction

Assume two binary relation symbols A and B and compute

$$x_1, x_3, x_2, x_4 \vdash A(x_1, x_2) \wedge B(x_3, x_4)$$

Example 1

`A.csv`	`B.csv`	Expected output
Pawel , Sobocinski	Julian , Rathke	Pawel , Julian , Sobocinski , Rathke

Example 2

`A.csv`	`B.csv`	Expected output
		1 ,3 ,2 ,4
1 ,2	3 ,4	1 ,3 ,2 ,4
1 ,2	3 ,4	1 ,3 ,2 ,4
		1 ,3 ,2 ,4

Example 3

`A.csv`	`B.csv`	Expected output
empty	pesto , genovese	empty

The order of variables to the left of $\vdash$ determines the order of the values in the output rows, as shown by Example 1. The second example is a demonstration that we are using what's usually called the *multi-relational* semantics (also used in SQL). The variables x_1, x_2 bind to values each row of the database A, while x_3, x_4 to those of B. Since for each row of A there are two rows of B, all possible four combinations are returned. In Example 3, the database A is empty, so there are no possible assignments to x_1, x_2: for this reason, the overall output is empty.

Problem 2 - Conjunction and variable repetition

Assume two binary relation symbols A and B and compute

$$x_1, x_2, x_3 \vdash A(x_1, x_2) \wedge B(x_2, x_3)$$

Example 1

A.csv	B.csv	Expected output
Sofiane , Boufal Dusan , Tadic Guido , Carillo	Boufal , Maserati Carillo , Ferrari	Guido , Carillo , Ferrari Sofiane , Boufal , Maserati

Example 2

A.csv	B.csv	Expected output
1 ,3 1 ,2	3 ,1 3 ,2 2 ,2	1 ,2 ,2 1 ,3 ,1 1 ,3 ,2

Example 3

A.csv	B.csv	Expected output
1 ,2 1 ,2	2 ,1 2 ,1	1 ,2 ,1 1 ,2 ,1 1 ,2 ,1 1 ,2 ,1

Here variables x_1, x_2 are determined by each row of A. Since we are using x_2 also in B, we are interested in only those rows of B for which the first value is equal to the value of x_2 in A. Note that the output is sorted lexicographically: for this reason the row starting with "Guido" comes before the row starting with "Sofiane" in the expected output of Example 1. Example 2 shows that a single row of A can "match" with more than one row of B. Example 3 is another demonstration of the use of multi-relational semantics.

Problem 3 - Equality

Assume two unary relation symbols P and Q and compute

$$x_1, x_2 \vdash P(x_1) \wedge Q(x_2) \wedge x_1 = x_2$$

Example 1

P.csv	Q.csv	Expected output
1 2 3 4	1 2 2 Sofa	1 ,1 2 ,2 2 ,2

Example 2

P.csv	Q.csv	Expected output
Alice Bob	Bob Alice	Alice , Alice Bob , Bob

Example 3

P.csv	Q.csv	Expected output
1 2	3 4	empty

Here x_1 ranges over P and x_2 over Q, but we are only interested in those assignments where the values agree. In Example 2, there are two such compatible bindings: the reason Alice,Alice comes before Bob,Bob is due to the lexicographical ordering of the output rows.

Problem 4 - Existential quantification

Assume a single binary relation symbol R and compute

$$x_1 \vdash \exists z.R(x_1, z)$$

Example 1

`R.csv`	Expected output
Michelangelo , Buonarotti	Antonello
Leonardo ,Da Vinci	Leonardo
Antonello ,Da Messina	Michelangelo

Example 2

`R.csv`	Expected output
Julian , Rathke	Jolion
Julian , Draxler	Julian
Julian , Assange	Julian
Jolion , Maugham	Julian

Here we list those bindings of x_1 to the first value in each row of R for which there is some z. We don't care what the z is and we don't list it. The output is ordered lexicographically.

Problem 5 - Existential quantification and conjunction

Assume that A and B are binary relation symbols and compute

$$x_1, x_2 \vdash \exists z.A(x_1, z) \wedge B(z, x_2)$$

Example 1

`A.csv`	`B.csv`	Expected output
Sofiane , Boufal	Boufal , Maserati	Guido , Ferrari
Dusan , Tadic	Carillo , Ferrari	Sofiane , Maserati
Guido , Carillo		

Example 2

`A.csv`	`B.csv`	Expected output
1 ,3	3 ,1	1 ,1
1 ,2	3 ,2	1 ,2
	2 ,2	1 ,2

Example 3

`A.csv`	`B.csv`	Expected output
		1 ,1
1 ,2	2 ,1	1 ,1
1 ,2	2 ,1	1 ,1
		1 ,1

The outputs for this problem are similar to those of Problem 2, but the existentially quantified variable is no longer free, does not appear to the left of the turnstile $\vdash$, and is therefore *not* outputted. Example 2, where the inputs are the same as the inputs in Example 2 of Problem 2, shows that this can affect the order in which outputs are given, since they must—as usual—be sorted in lexicographical order.

First submission - due Thursday March 15 4pm

You will be required to submit a zip file containing:

- the sources for the interpreter for your language, written in Haskell
- five programs, written in **YOUR** language, that solve the five problems specified above. The programs should be in files named `pr1.cql`, `pr2.cql`, `pr3.cql`, `pr4.cql`, `pr5.cql`.

We will compile your interpreter using the command `make` on linuxproj, so you will need to include a Makefile in your zip file that produces an executable named `myinterpreter`. Prior to submission, you are required to make sure that your interpreter compiles **on linuxproj**: if your code does not compile then you will be awarded 0 marks. Before submission, we will release a simple "automarking" script that will allow you to check if your code compiles and whether each of your programs passes a basic test.

Your interpreter is to take a file name (that contains a program in your language) as a single command line argument. The interpreter should produce any output on standard output (`stdout`) and any error messages on standard error (`stderr`).

For each problem, we will test whether your code performs correctly by using a number of tests. We only care about correctness and performance will not be assessed (within reason). You can assume that for the tests we will use correctly formatted input.

For example, when assessing your solution for problem 1 we will run

```
myinterpreter pr1.cql
```

in a directory where we also provide our own versions of `A.csv` and `B.csv`.

All submissions will be done through handin. The precise submission instructions will be released a few days before the deadline. The marks awarded for the first submission will count 20% of the total coursework mark (4% for each of the five problems). You will receive feedback based on the our testing prior to the second deadline.

Second submission - due Thursday April 26 4pm

Shortly after the first deadline we will release a further five problems. Although they will be different from Problems 1-5, you can assume that they will be closely related, and follow the same input/output conventions. You will be required to submit two separate files.

First, you will need to submit a zip file containing programs (`pr6.cql`, `pr7.cql`, `pr8.cql`, `pr9.cql`, `pr10.cql`) written in your language that solve the additional problems. We will run our tests on your solutions and award marks for solving the additional problems correctly. This will form 50% of the total coursework mark (10% each for the five additional problems). You will have the option of resubmitting the interpreter, for a 50% penalty on this component. Thus, if you decide to resubmit your interpreter in the second submission the maximum possible total coursework mark is capped at 75%.

Second, you will be required to submit a 3 page user manual for your language in pdf format that explains the syntax, and describes any additional features (programmer convenience, type checking, informative error messages, etc.) This report, together with the five programs will be evaluated qualitatively and your marks will be awarded for the elegance and flexibility of your solution and the clarity of the user manual. These qualitative aspects will be worth 30% of the total coursework mark.

As you know, the coursework is to be done in pairs. As part of the second submission we will require a declaration of how marks are to be distributed amongst the members of your group. You will receive all feedback and your marks by Thursday May 17.