MIF14-BDD: Project
Implementing a Top-down Query Evaluation Engine for Datalog

May 14, 2021

Abstract

This project will let you delve into the details of the implementing a query evaluation engine for Datalog. The project consists of two main parts.
The first part (about 12 points) consists of **implementing** an engine that, given as input a set of facts, a set of rules and a query, computes the answers, through a *top-down* evaluation heuristic.
The second part (about 8 points) consists of **analyzing** the performance of your engine.
To this end, you should:

- (about 4 points) *benchmark*, i.e., write a set of 10 example Datalog programs and queries to be used as sample inputs, together with their respective translations into DES syntax;
- (about 2 points) *test*, i.e., run your engine on the benchmark you designed and check that it outputs the same result as DES;
- (about 2 points) *evaluate*, i.e., compare the runtimes you obtain against those of DES and/or of other open-source Datalog engines and plot the results.

This project is to be performed by one or two students. The programming language is `Java`. A small report (at most 2 pages, without appendix) must be added to your project.

The project is due on June 10, 2021 at noon (12pm).
You have to submit it on Tomuss by this deadline (no email will be accepted).

1 The Datalog Language

A Datalog clause is of the form:

$$\forall \overline{x}. \forall \overline{y}. (\phi(\overline{x}, \overline{y}) \rightarrow H(\overline{y})).$$

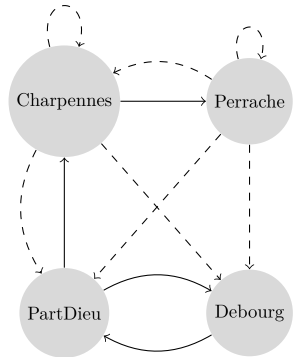
The formula $\phi(\overline{x}, \overline{y})$ is built from the conjunction of positive relational atoms having $\overline{x}$ and $\overline{y}$ as free variables. In the remainder, quantifiers and term arguments will be omitted.
A clause C clause is thus of the form: $A_1, \dots, A_n \rightarrow H$, where $head(C) = H$, $body(C) = \{A_1, \dots A_n\}$.

The following safety condition is imposed on all clauses of a program P:

all head variables should appear among those in the body.

Example 1.1 (Reachability Queries)**.** Consider the following Datalog program. It consists of a set of **facts**, denoting that there is a *link* — marked by an arrow — between two stations, and of a set of **rules** indicating the *metro* names and the *reachability* between two stations (via multiple links) — marked by a dotted arrow. The program **query** computes all *metro* stations that are *reachable* from *Charpennes*.

```
link(Charpennes, Perrache)
link(PartDieu, Charpennes)
link(Debourg, PartDieu)
link(PartDieu, Debourg)
link(X,Y)→metro(X).
link(X,Y)→metro(Y).
link(X,Y)→reachable(X,Y).
link(X,Z), reachable(Z,Y)→reachable(X,Y).
reachable(Charpennes, Y) → query(Y).
```



The *bottom-up* evaluation of the program above, proceeds as follows to compute its fixpoint model:

$\mathbf{T^0_P}(\emptyset) \; = \{link(Charpennes, Perrache), link(PartDieu, Charpennes), link(Debourg, PartDieu), link(PartDieu, Debourg)\}$

$\mathbf{T^1_P}(T^0_P) = T^0_P \cup \{metro(Charpennes), metro(Perrache), metro(PartDieu), metro(Debourg), reachable(Charpennes, Perrache),$ $reachable(PartDieu, Charpennes), reachable(Debourg, PartDieu), reachable(PartDieu, Debourg)\}$

$\mathbf{T^2_P}(T^1_P) = T^1_P \cup \{reachable(PartDieu, Perrache), reachable(Debourg, Charpennes), reachable(Debourg, Debourg),$ $reachable(PartDieu, PartDieu), \mathbf{query(Perrache)}\}$

$\mathbf{T^3_P}(T^2_P) = T^2_P \cup \{reachable(Debourg, Perrache)\}$

Note that this heuristic is *sub-optimal*, as the engine infers additional reachability facts that are *not relevant* for computing the needed query. In particular, as this example illustrates, it is sometimes more efficient to "push" the constants from the query definition (i.e., $Charpennes$) down into the rules that are relevant for the query, i.e., those deriving into the predicates being queried. This creates more (sub)queries from the atoms belonging to the bodies of these rules. In turn, subqueries are answered in a similar, "top-down" fashion. We will implement such *top-down* heuristic, namely the **Query-Subquery (QSQ)** algorithm.

2 Example Top-Down Query-Subquery Evaluation

The idea of **QSQ** evaluation is to *unify* the distinguished query atom, i.e., $reachable(Charpennes, Y)$, with the *relevant* rule heads and then propagate the acquired information inside the respective rule bodies. Each body atom becomes a subquery that is answered similarly, by unifying them with *relevant* rules, possibly resulting in more subqueries. When all subqueries pertaining to a rule body are answered, **QSQ** produces an answer set for the (sub)query pertaining to the rule head. The production and answering of queries/subqueries repeat until no more answer sets and no more subqueries are obtained.

We explain this procedure on the previous example. For the $reachable(Charpennes, Y)$ query, the *relevant* rules from our previous program are R_1 and R_2 given below.

$R_1 \quad link(X, Y) \rightarrow reachable(X, Y)\,.$
$R_2 \quad link(X, Z), reachable(Z, Y) \rightarrow reachable(X, Y)\,.$

Top-Down Information Passing ↓ Passing the variable binding information to a rule head is called *top-down information passing*. In particular, we seek to determine what are the constants with which we can replace head variables. Let us understand how this works by focusing on R_1.

$R_1 \quad link(X, Y) \rightarrow reachable(X, Y)\,.$

Unifying the query atom $reachable(Charpennes, Y)$ with the head of the first rule below, $reachable(X, Y)$, first introduces the variable binding $\{X \mapsto Charpennes\}$. In this case, we say that the variable X is *bound*, as we have found a constant with which we can instantiate it.

Sideways Information Passing $\leftarrow$ Passing the variable binding information from atom to atom in the same rule body is called *sideways information passing*. "Pushing" the variable binding information $\{X \mapsto Charpennes\}$ into the body of R_1, we obtain:

R_1 $link(Charpennes, Y) \rightarrow reachable(Charpennes, Y)$.

We can now take as a subquery the atom $link(Charpennes, Y)$. As $link(Charpennes, Perrache)$ is in our set of facts, we obtain the binding information $\{Y \mapsto Perrache\}$. Hence, we now know that $reachable(Charpennes, Perrache)$.

Let us illustrate the $\downarrow$ and $\leftarrow$ information passing steps on rule R_2 as well.

R_2 $link(X, Z), reachable(Z, Y) \rightarrow reachable(X, Y)$.

From the $\downarrow$ step, we obtain, as previously, $\{X \mapsto Charpennes\}$. Applying a $\leftarrow$ step, we get:

R_2 $link(Charpennes, Z), reachable(Z, Y) \rightarrow reachable(Charpennes, Y)$.

We now take $link(Charpennes, Z)$ to be our subquery. As before, since $link(Charpennes, Perrache)$ is in our set of facts, we obtain the binding information $\{Z \mapsto Perrache\}$. We can now *backtrack* and continue to propagate the information sideways in another $\leftarrow$ step. We thus have:

R_2 $link(Charpennes, Perrache), reachable(Perrache, Y) \rightarrow reachable(Charpennes, Y)$.

We now take $reachable(Perrache, Y)$ to be our subquery. We repeat the same procedure. The *relevant* rules are still R_1 and R_2. Consider R_1:

R_1 $link(X, Y) \rightarrow reachable(X, Y)$.

From the $\downarrow$ step, we get $\{X \mapsto Perrache\}$. Applying the $\leftarrow$ step:

R_1 $link(Perrache, Y) \rightarrow reachable(Perrache, Y)$.

When taking $link(Perrache, Y)$ to be our subquery, we notice there is no fact it can be unified against, so no new facts can be inferred using R_1. We *backtrack* to the $reachable(Perrache, Y)$ subquery and are left to consider R_2:

R_2 $link(X, Z), reachable(Z, Y) \rightarrow reachable(X, Y)$.

From the $\downarrow$ step, we get $\{X \mapsto Perrache\}$. Applying the $\leftarrow$ step:

R_2 $link(Perrache, Z), reachable(Z, Y) \rightarrow reachable(Perrache, Y)$.

When taking $link(Perrache, Z)$ to be our subquery, we notice there is no fact it can be unified against, so no new facts can be inferred using R_2 either. Hence, no new reachability information can be obtained.

3 Subquery Components

To be able to process a subquery we need two ingredients:

- *argument information* – what are the *bound* variables that should be replaced by constants ?
- *binding information* – what is the *set of substitutions* that we can pass $\downarrow$ or $\rightarrow$?

Argument Information To explicitly mark which arguments are ready to receive binding information, for every **IDB predicate** P of our program, we compute its *adorned* version P^γ. The parameter γ is a list, whose length corresponds to the *arity* of P and whose elements are boolean flags, denoted b and f, marking whether the predicate argument in the respective position is $b$ound or $f$ree.

Adornments propagate to rules, as every rule body atom B_i is rewritten to refer to an *adorned* predicate. To compute an adornment, we determine if the argument at each position is bound, by checking if:

- it is a *constant*
- it is a variable bound in the rule head, according to the adornment of the head atom
- it is a variable bound by some other atom B_j, where $j < i$

Note that different atom orderings in the rule body can yield different adornments.

For example, the *adornment* of the rules we considered previously is:

R_1 $link(X, Y) \rightarrow reachable^{bf}(X, Y)$.
R_2 $link(X, Z), reachable^{bf}(Z, Y) \rightarrow reachable^{bf}(X, Y)$.
R_3 $reachable^{bf}(Charpennes, Y) \rightarrow \mathbf{query^f(Y)}$.

- In R_3, since the first body atom has a constant argument the correspoding adornment is $reachable^{bf}$, marking that the first position is *bound*, while the second position contains the *free* variable Y.
- In R_2, since X is bound in the head, it is also bound in the first argument of the first body atom; also, since any match for the first body atom bounds Z, we have that Z is also bound when evaluating the second atom, due to the left-to-right evaluation.
- In R_1, since X is bound in the head, it is also bound in the first argument of the first body atom.

Binding Information/Auxiliary Relations A *auxiliary relation* R consists of the set of all possible substitutions for the *bound arguments* of an adorned predicate P^γ and represents the *binding information* to be passed $\downarrow$ or $\leftarrow$. For example, $R = \{\{X \mapsto Charpennes, Y \mapsto Perrache\}\}$ contains a possible binding for X and Y.

Depending on the direction in which its bindings will applied, R can either be:

- an *input relation* – $R \triangleq input\_P^\gamma(V)$ – representing information passed $\downarrow$ into the head of adorned rules deriving P^γ, where V **is a set of bound arguments** in P^γ and the *arity* of $input\_P^\gamma(V)$ is equal to the number of b arguments in γ
- a *supplementary relation* – $R \triangleq sup^i_j(V)$ – representing information passed $\leftarrow$ into the j-th atom of the i-th rule, where V is a set of arguments that are:
  - bound by atoms before j
  - later referenced
- a *output relation* – $R \triangleq output\_P^\gamma(V)$ – representing the "completed" input, with added values for all unbound variables added, where the *arity* of $output\_P^\gamma(V)$ is equal to the length of γ

Note that when evaluating the body atoms from left to right, in a given rule i, with head predicate P, the first set of bindings sup^i_0 come from $input\_P^\gamma$ – namely, they are the set of substitutions for *the* ***bound*** *variables in the head* – and the last set of bindings sup^i_n – namely, the set of substitutions for ***all*** *variables in the head* – go to $output\_P^\gamma$.

For example, we mark the auxiliary relations in the previously considered rules as follows:

R_1 $[sup^1_0(X)]\ link(X, Y)\ [sup^1_1(X, Y)] \rightarrow reachable^{bf}(X, Y)\ [output\_reachable^{bf}(X, Y)]$.
R_2 $[sup^2_0(X)]\ link(X, Z),\ [sup^2_1(X, Z)]\ reachable^{bf}(Z, Y)\ [sup^2_2(X, Y)] \rightarrow reachable^{bf}(X, Y)\ [output\_reachable^{bf}(X, Y)]$.
R_3 $[input\_reachable^{bf}(X)]\ reachable^{bf}(Charpennes, Y) \rightarrow \mathbf{query^f(Y)}$.

4 QSQR Evaluation Steps

Preprocessing and Initialization Before the evaluation stage, we first add the needed *argument and binding information* to our program.

- We create the *adorned program* by recursively creating adorned rules for all adorned predicates.
- We initialize all *auxiliary relations* to empty sets.

The QSQR evaluation procedure consists of the following four steps:

Step 1 The query atom – with predicate P – is *unified* with the *relevant* adorned rules – with head predicates P^{γ}, according to which of its arguments are *bound*. The corresponding substitutions are added to the input relation, namely $input\_P^{\gamma}$.

Step 2 For a *relevant* adorned rule i, we compute $supp_0^i$, by projecting out the necessary variables from the input relation.

Step 3 We produce and evaluate subqueries by computing subsequent *auxiliary relations*. For a *relevant* adorned rule i – with body $B \triangleq A_1, \dots, A_l$, we do the following:

- if the predicate S of A_k is an EDB, given $supp_{k-1}^i$, we compute $supp_k^i$, by adding to each set of substitutions in $supp_{k-1}^i$, corresponding bindings that instantiate the atom's unbound variables to the appropriate constants of the facts with predicate S.
- if the predicate S^{γ_k} of A_k is an IDB, we add to $input\_S^{\gamma_k}$, new bindings from $supp_k^i$ combined with constants in A_k
- if $input\_S^{\gamma_k}$ changed, *recursively* evaluate the subqueries corresponding to all rules with head predicate S^{γ_k}
- compute $supp_k^i$ from $supp_{k-1}^i$ and $output\_S^{\gamma_k}$.

Step 4 Add the results of $supp_l^i$ to $output\_P^{\gamma}$.

We illustrate the evaluation, by revisiting our running example, in which we have our query relation, with the input relation explicitly marked:

R_3 $[input\_reachable^{bf}(X)]$ $reachable^{bf}(Charpennes, Y) \rightarrow \mathbf{query^f(Y)}$.

Step 1 We compute $input\_reachable^{bf}(X)$, by *unifying* the query body atom $reachable^{bf}(Charpennes, Y)$, against the adorned head $reachable^{bf}(X, Y)$ of the relevant rule R_1.

R_1 $link(X, Y) \rightarrow reachable^{bf}(X, Y)$.

We obtain: $input\_reachable^{bf}(X) = \{\{X \mapsto Charpennes\}\}$.

Step 2 We then compute $sup_0^1(X)$, by projecting from $input\_reachable^{bf}(X)$ the bindings corresponding to the variables marked as bound in the adornment of the head predicate ($reachable^{bf}$), i.e., for X. Hence, $sup_0^1(X) = \{\{X \mapsto Charpennes\}\}$.

R_1 $[sup_0^1(X)]$ $link(X, Y) \rightarrow reachable^{bf}(X, Y)$.

Step 3 Next, we compute $sup_1^1(X, Y)$. To this end, we notice that $link$ is a EDB predicate and that we can directly compute the binding $\{Y \mapsto Perrache\}$ from $sup_0^1(X, Y)$.
Hence, $sup_1^1(X, Y) = \{\{X \mapsto Charpennes, Y \mapsto Perrache\}\}$.
It follows that $output\_reachable(X, Y) = \{\{X \mapsto Charpennes, Y \mapsto Perrache\}\}$

and $sup_0^2(X)$, by projecting from $input\_reachable^{bf}(X)$ the bindings corresponding to the variables marked as bound in the adornment of each of the head predicates in R_1 and R_2. In our setting, this bound variable corresponds to X in both cases and $sup_0^1(X) = sup_0^2(X) = \{\{X \mapsto Charpennes\}\}$.

R_1 $[sup_0^1(X)]$ $link(X, Y) \rightarrow reachable^{bf}(X, Y)$.
R_2 $[sup_0^2(X)]$ $link(X, Z), reachable^{bf}(Z, Y) \rightarrow reachable^{bf}(X, Y)$.

In R_1, we use the *binding information* provided by $sup_0^1(X)$ to
We obtain that $input\_reachable^{bf}(X) = \{\{X \mapsto Charpennes\}\}$, $sup_0^1 = sup_0^2 = \{X \mapsto Charpennes\}$.

5 Input File Format

In the first part, we assume an input file containing a stratified Datalog programs in textual form. Their syntax is specified in BNF in Annex A.1. A parser (using JavaCC [1]) that checks the syntactic correctness of the program, according to the provided BNF grammar is available on Google drive for your convenience (you do not need to implement it).

Note that for handling constants in the atoms of a mapping, we have to declare a special **unique**, unary atom for that constant, as illustrated in the below example, i.e., to express:

```
EDB
link(Charpennes,Perrache)
link(PartDieu,Charpennes)
link(Debourg,PartDieu)
link(PartDieu,Debourg)

IDB
metro($x)
reachable($x,$y)
query($y)


MAPPING
reachable(Charpennes,$y) -> query($y).
```

...we will write:

```
EDB
link(Charpennes,Perrache)
link(PartDieu,Charpennes)
link(Debourg,PartDieu)
link(PartDieu,Debourg)
cst(Charpennes)


IDB
metro($x)
reachable($x,$y)
query($y)


MAPPING
cst($x), reachable($x,$y) -> query($y).
```

[1] https://javacc.java.net/

Example 5.1. The full Example 1.1 is written according to the BNF grammar as follows:

```
EDB
link(Charpennes, Perrache)
link(PartDieu, Charpennes)
link(Debourg, PartDieu)
link(PartDieu, Debourg)
cst(Charpennes)

IDB
metro($x)
reachable($x, $y)
query($y)

MAPPING
link($x, $y) -> metro($x).
link($x, $y) -> metro($y).
link($x, $y) -> reachable($x, $y).
link($x, $z), reachable($z, $y) -> reachable($x, $y).
cst($x), reachable($x, $y) -> query($y).
```

A Formal Grammar

A.1 BNF for stratified Datalog clauses

The BNF for input files is given below. Note that, for sake of simplicity, usual skippable characters (spaces, tabulations, carriage returns) have been omitted. Comments (also omitted and to be skipped) start with two dashes (`--`) and finish at the end of the line.

```
START      ::= "EDB" SCHEMA "IDB" SCHEMA "MAPPING" TGDS

SCHEMA     ::= RELATION SCHEMA
             | RELATION

RELATION   ::= NAME "(" ATTS ")"

ATTS       ::= NAME "," ATTS
             | NAME

TGDS       ::= TGD
             | TGDS TGD

TGD        ::= QUERY "->" ATOM "."

LITERAL    ::= "NEG" ATOM
             | ATOM

QUERY      ::= LITERAL "," QUERY
             | LITERAL

ATOM       ::= NAME "(" ARGS ")"

ARGS       ::= VALUE "," ARGS
             | VALUE

VALUE      ::= VARIABLE
             | CONSTANT

VARIABLE   ::= "$" NAME

NAME       ::= LETTER
             | LETTER NAME2

NAME2      ::= LETTER_OR_DIGIT
             | LETTER_OR_DIGIT NAME2

CONSTANT   ::= DIGITS

DIGITS     ::= DIGIT DIGITS
             | DIGIT

LETTER_OR_DIGIT ::= LETTER
                  | DIGIT

LETTER     ::= "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j"
             | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t"
             | "u" | "v" | "w" | "x" | "y" | "z"
             | "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J"
             | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T"
             | "U" | "V" | "W" | "X" | "Y" | "Z"
             | "_"

DIGIT      ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```