MIF14-BDD: Project
Implementing and Testing a Stratified Datalog Engine
Part I: A Stratified Datalog Engine in Java

April 14, 2017

Abstract

This project will let you delve into the details of the implementing and evaluating a stratified Datalog engine. The project consists of two main parts.
The first part (about 12 points) consists of **implementing** an engine that computes all the facts that can be infered from a stratified Datalog program, given as input.
The second part (about 8 points) consists of **testing** the performance of your engine. To this end, you should check the correctness of its output by comparing it with that obtained by **either**:

- (about 2 points) *manually* translating your program into DES **and** SQL and evaluating using both DES **and** Oracle
- (about 8 points) writing an *automatic* translation of your program to SQL with recursion and evaluating using Oracle

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 May 21, 2017 at 11pm.
You have to submit it on Spirals by this deadline (no email is accepted).
In order to submit, you have to go here:
http://spiralconnect.univ-lyon1.fr/webapp/activities/activities.jsp?containerId=6689641

1 Stratified Datalog

A *stratified* 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/negative 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:

$$B_1, \dots, B_k, NEG\ B_{k+1}, \dots, NEG\ B_n \rightarrow H.$$

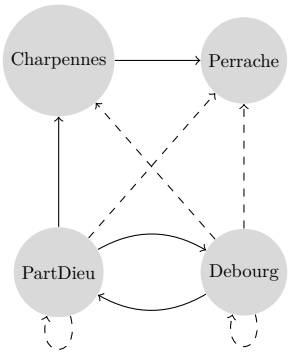
We denote $head(C) = H$, $body(C) = \{B_1, \dots B_n\}$, $body^+(C) = \{B_1, \dots, B_k\}$ and $body^-(C) = \{B_{k+1}, B_n\}$. Also we call *EDB* relations those that appear among the program facts (clauses with no body) and *IDB* relations, the rest of the relations in the program.

The following safety conditions are usually imposed on all clauses of a *stratified* program P:

- Safety Condition: all head variables should appear among those in the body
- Extended Safety Condition: all variables that appear in the negative body atom should also appear in a positive one

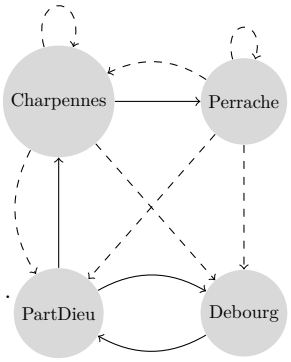
Example 1.1 (Positive Datalog)**.** Consider the following *positive* Datalog program. Given a set of facts, denoting that there is a *link* between two stations, it infers the *metro* names and *reachability* between two stations (via multiple links).

```
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).
```



Example 1.2 (Stratified Datalog)**.** Let us now consider the *stratified* Datalog program below. Given a set of facts, denoting *metro* stations and *reachability* between them, it infers the *unreachability* between two metro stations. We obtain the program below.

```
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).
metro(Y), metro(Y), NEG reachable(X,Y) → unreachable(X,Y).
```



Example 1.3 (Semipositive Datalog)**.** The following program also infers *unreachability*. It however has the particuliarity that the relation of the negated atom is an EDB relation. Such programs are called *semipositive*, as they are equivalent to *positive* ones.

```
metro(Charpennes)
metro(Perrache)
metro(PartDieu)
metro(Debourg)
reachable(Charpennes, Perrache)
reachable(PartDieu, Charpennes)
reachable(Debourg, PartDieu)
reachable(PartDieu, Debourg)
reachable(PartDieu, Perrache)
reachable(Debourg, Charpennes)
reachable(Debourg, Debourg)
reachable(PartDieu, PartDieu)
reachable(Debourg, Perrache)
metro(X), metro(Y), NEG reachable(X,Y)→unreachable(X,Y).
```

Example 1.4 (From Semipositive to Positive Datalog)**.** Indeed, the program in Example 1.3 is equivalent to the following one, in which the relation `reachable` is replaced by its *complement* `creachable`:

```
metro(Charpennes)
metro(Perrache)
metro(PartDieu)
metro(Debourg)
creachable(Charpennes, Charpennes)
creachable(Charpennes, PartDieu)
creachable(Charpennes, Debourg)
creachable(Perrache, Charpennes)
creachable(Perrache, PartDieu)
```

```
creachable(Perrache,Perrache)
creachable(Perrache,Debourg)
metro(X),metro(Y),creachable(X,Y)→unreachable(X,Y).
```

2 Stratified Datalog: Stratification and Partitioning

Let P be a program with clauses C of the form:

$$B_1, \ldots, B_k, NEG\ B_{k+1}, \ldots, NEG\ B_n \rightarrow H.$$

We say a relation symbol is *defined* if it appears in the head of a clause in P.

Stratification of Program Symbols A stratification mapping σ indexes relation symbols in P, such that:

1) symbols of positive atoms are defined earlier than/simultaneously with the symbol of the head atom $\sigma(sym(B_j)) \leq \sigma(sym(H))$, for $j \in [1, n]$

2) symbols of negated atoms are defined **strictly before** the symbol of the head atom $\sigma(sym(B_j)) < \sigma(sym(H))$, for $j \in [k+1, n]$

Computing a Stratification There are multiple valid ways to stratify P.
One such stratification can be obtained through the below algorithm:

1: **procedure** STRATIFICATION(P)
2: **for all** predicates p in P **do**
3: $\text{stratum}[p] \leftarrow 1$
4: **repeat**
5: **for all** clause C in P with head predicate p **do**
6: **for all** negated subgoal of C with predicate q **do**
7: $\text{stratum}[p] \leftarrow \max(\text{stratum}[p], 1 + \text{stratum}[q])$
8: **for all** nonnegated subgoal of C with predicate q **do**
9: $\text{stratum}[p] \leftarrow \max(\text{stratum}[p], \text{stratum}[q])$
10: **until** no stratum changes or a stratum exceeds the predicate count in P

Example 2.1 (Program Stratification)**.** Consider the program P consisting of the following clauses:

$$\begin{aligned}
C_1 &= NEG\ q(X), r(X) \rightarrow p(X) \\
C_2 &= NEG\ t(X), q(X) \rightarrow p(X) \\
C_3 &= s(X), NEG\ t(X) \rightarrow q(X) \\
C_4 &= t(X) \rightarrow r(X) \\
C_5 &= q(a). \\
C_6 &= s(b). \\
C_7 &= t(a).
\end{aligned}$$

After iterating the stratification algorithm twice, each predicate symbol in P is associated to a stratum number, as illustrated in the last line of the table below :

	p	q	r	s	t
	1	1	1	1	1
C_1	2	1	1	1	1
C_2	2	1	1	1	1
C_3	2	2	1	1	1
⋮	⋮	⋮	⋮	⋮	⋮
C_7	2	2	1	1	1
C_1	3	2	1	1	1
⋮	⋮	⋮	⋮	⋮	⋮
C_7	**3**	**2**	**1**	**1**	**1**

Computing a Program Partitioning Let P be a stratified Datalog program P and σ a stratification.

Symbol Definitions. The *definition* $def(s)$ of a program symbol s is the set of all program clauses that have the symbol p in their head atom.
Program Slices. Each *slice* P_i of P collects all symbol definitions $def(s)$ for symbols s mapped to i by σ.
Program Partitioning. The stratification σ induces a partitioning of P into corresponding slices.

Example 2.2 (Program Partitioning)**.** Revisiting Example 2.1, $def(p) = \{C_1, C_2\}$ and $def(q) = \{C_3, C_5\}$.
$P_1 = def(r) \cup def(s) \cup def(t) = \{C_4\} \cup \{C_6\} \cup \{C_7\} = \{C_4, C_6, C_7\}$.
$P_2 = def(q) = \{C_3, C_5\}$
$P_3 = def(p) = \{C_1, C_2\}$
Hence, $P = P_1 \sqcup P_2 \sqcup P_3$, where, as computed above:

$$P_1 = \left\{ \begin{array}{l} t(X) \rightarrow r(X) \\ s(b). \\ t(a). \end{array} \right\}, \; P_2 = \left\{ \begin{array}{l} s(X), NEG\ t(X) \rightarrow q(X). \\ q(a). \end{array} \right\}, \; P_3 = \left\{ \begin{array}{l} NEG\ q(X), r(X) \rightarrow p(X). \\ NEG\ t(X), q(X) \rightarrow p(X). \end{array} \right\}$$

3 Evaluating a Positive Datalog Program

Let P be a *positive* Datalog program, $adom(P)$ the set of program **constants** (the active domain) and $Var(P)$ the set of program **variables**.
Groundings A *grounding* ι is a mapping $\iota : Var(P) \rightarrow adom(P)$.
Clause Instantiation For a clause C in P of the form $p_1(\vec{t_1}), \ldots, p_m(\vec{t_m}) \rightarrow p_0(\vec{t_0})$, the *clause instantiation* ιC is $p_1(\iota\vec{t_1}), \ldots, p_m(\iota\vec{t_m}) \rightarrow p_0(\iota\vec{t_0})$.

Example 3.1 (Clause Instantiation)**.** Let us revisit Example 1.1. Consider the grounding ι, such that $\iota X =$ `PartDieu`, $\iota Y =$ `Charpennes` and $\iota Z =$ `Debourg` and the clause:
`link(X,Z),reachable(Z,Y)` $\rightarrow$ `reachable(X,Y)`. The corresponding clause instantiation is:
`link(PartDieu,Debourg),reachable(Debourg,Charpennes)` $\rightarrow$ `reachable(PartDieu,Charpennes)`.

The evaluation of a program P proceeds clause by clause. For each such clause C, it aims to construct a grounding ι, such that **all** the *body* atoms in the instantiation ιC are in the existing facts. If this is the case, it adds a *new* fact, namely the *head* of ιC. The procedure repeats until no new facts can be added.

```
1:  function EVALUATION(P)
2:      M ← ∅
3:      I_0 ← ∅
4:      repeat
5:          for all clauses C in P do
6:              for all groundings ι : Var(C) → adom(P) do
7:                  if all atoms in body(ιC) ∈ I_j then
8:                      I_{j+1} ← I_j ∪ head(ιC)
9:      until I_{n+1} = I_n
10:     M ← I_{n+1}
11:     return M
```

Example 3.2 (Program Evaluation)**.** For program P_1 in Example 2.2, we have that $I_1 = \{s(b), t(a)\}$ and, for $\iota : X \mapsto a$, $\iota(t(X)) \in I_1$. Hence, $I_2 = \{s(b), t(a), r(a)\}$. As no further facts can be added, $M = I_2$.

Example 3.3 (Program Evaluation)**.** For the program in Example 1.1, we have that:

$I_0 = \emptyset$
$I_1 = \{link(Charpennes, Perrache), link(PartDieu, Charpennes), link(Debourg, PartDieu), link(PartDieu, Debourg)\}$
$I_2 = I_1 \cup \{reachable(Charpennes, Perrache), reachable(PartDieu, Charpennes), reachable(Debourg, PartDieu), reachable(PartDieu, Debourg)\}$
$I_3 = I_2 \cup \{reachable(PartDieu, Perrache), reachable(Debourg, Charpennes), reachable(Debourg, Debourg), reachable(PartDieu, PartDieu)\}$
$I_4 = I_3 \cup \{reachable(Debourg, Charpennes)\}$
$I_5 = I_4$

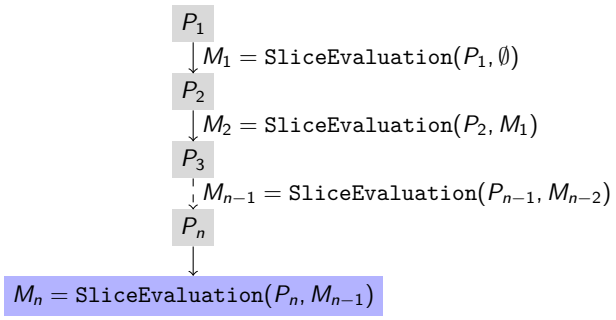
Hence $M = I_5$

4 Evaluating a Stratified Datalog Program

Let P be a *stratified* Datalog program, σ a stratification and $P_1, \ldots, P_n$ the partitioning slices. Stratifications (see Section 2) ensure *each* program slice in the induced partitioning is *semipositive*. Hence, each slice can be evaluated *independently* with an algorithm similiar to that in Section 3. This is given below.

```
1:  function SliceEvaluation(P_i, M_{i-1})
2:      M_i ← M_{i-1}
3:      I_0 ← ∅
4:      repeat
5:          for all clauses C in P do
6:              for all groundings ι : Var(C) → adom(P) do
7:                  if all atoms in body⁺(ιC) ∈ I_j and all atoms in body⁻(ιC) ∉ I_j then
8:                      I_{j+1} ← I_j ∪ head(ιC)
9:      until I_{n+1} = I_n
10:     M_i ← M_i ∪ I_{n+1}
11:     return M_i
```

The *stratified* evaluation of P will iteratively evaluate each slice, from the bottom-up, as depicted.




```
1:  function StratifiedEvaluation(P_1, ..., P_n)
2:      M_0 ← ∅
3:      i ← 1
4:      repeat
5:          M_i ← SliceEvaluation(P_i, M_{i-1})
6:          i ← i + 1
7:      until i > n
8:      return M_n
```

Example 4.1 (Stratified Evaluation)**.** Let $P = \begin{cases} q(a).\ s(b).\ t(a).\ t(X) \to r(X). \\ NEG\ q(X), r(X) \to p(X). \\ NEG\ t(X), q(X) \to p(X). \\ s(X), NEG\ t(X) \to q(X). \end{cases}$ for which a *stratification* $\sigma(s) = 1$, $\sigma(t) = 1$, $\sigma(r) = 1$, $\sigma(q) = 2$, $\sigma(p) = 3$, with the *strata* $\sigma_1 = \{s, t, r\}$, $\sigma_2 = \{q\}$, $\sigma_3 = \{p\}$,

induces the partitioning $P = P_1 \sqcup P_2 \sqcup P_3$, with the *slices*:
$P_1 = \left\{ \begin{array}{l} s(b).\ t(a). \\ t(X) \rightarrow r(X). \end{array} \right.$ $P_2 = \left\{ \begin{array}{l} q(a). \\ s(X), NEG\ t(X) \rightarrow q(X). \end{array} \right.$ $P_3 = \left\{ \begin{array}{l} NEG\ q(X), r(X) \rightarrow p(X). \\ NEG\ t(X), q(X) \rightarrow p(X). \end{array} \right.$ We have:
$M_0 = \emptyset$
$M_1 = \texttt{SliceEvaluation}(P_1, M_0) = \{r(a), s(b), t(a)\}$
$M_2 = \texttt{SliceEvaluation}(P_2, M_1) = M_1 \cup \{q(a), q(b)\}$
$M_3 = \texttt{SliceEvaluation}(P_3, M_2) = M_2 \cup \{p(b)\} = \{r(a), s(b), t(a), q(a), q(b), p(b)\}$.

5 Java Implementation

5.1 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]) is provided on Spirals for your convenience (you do not need to implement it). It checks the syntactic correctness of your program and whether the Safety Condition in Section 1 is satisfied. Note that, for evaluating stratified Datalog programs, we do not need to impose that the Extended Safety Condition in Section 1 is satisfied.

Example 5.1. Example 1.3 is written according to the BNF grammar as follows:

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

IDB
metro($x)
reachable($x,$y)
unreachable($x,$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).
metro($y), metro($y), NEG reachable($x,$y) -> unreachable($x,$y).
```

5.2 Workflow

One the input text file containing a stratified Datalog program is parsed, you should:

- characterize the program: determine if it is positive, semipositive or stratified
- compute a corresponding symbol stratification
- compute a program slicing
- implement an evaluation algorithm, based on the pseudo-code provided in Section 4 and Section 3.

5.3 Remarks

- you are not obliged to follow the provided pseudo-code, *as long as your engine runs and correctly computes the expected output*
- additional points can be earned for optimized implementations

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

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"
```