CONTRERAS Baptiste
CLEMENT Florent

Stratification module for datalog programs

Table of Contents

1. Implementation choices

For this project we decided to use Python and Antlr4. Antlr4 is a parser. We use it to read the input file, parse it and check if there is any syntax error. If the syntax is good, Antlr4 give us a AST (Abstract Syntax Tree). We use this tree to build a pool of rules and predicates. At this point of the program execution, we have all the rules built as object in a list and we can apply the given algorithm on it. The last step is to take the output of the algorithm (which is a dict with the stratum number as key and a list of rules for this layer as value) and print it. We wanted to use Antlr4 because it enables us to create a complete and powerful syntax for the inputs of our program. We can recognize attributes of a predicate, we can determine if an attribute is a value (string like “toto” or int like 1). We are also able to apply some validation on the rules and inputs. the most import thing is that we can easily manipulate the predicates and rules as Python object. To sum up in a few words, our implementation works well and we can easily make it evolve.

2. How to use the program

python3 main.py path/to/file.txt

If there is no syntax error, the output is printed in the console.

3. Inputs syntax

%edb represent a edb block and %idb an idb block

Predicates and rules must be declared in their respective block.
Each program must contain both blocks.
Predicates must have their first letter in uppercase (like *Toto*, *Predicate*).
Uppercase is just allowed at the start of a predicate name (*ToTo* or *PreDiCate* are forbidden).
A predicate can have zero or multiple attributes (*Toto()* or *Toto(a,b,c)*).
An attribute can be:

- a variable like *id*, *name* . Uppercase is not allowed on the first character of a variable name. (*Toto(id,name)* is valid but *Toto(Id, Name)* isn't)
- a string value like *"M"*, *"toto"* . (for example : *Toto("M", "Hello world")*)
- an int value, like 1,2,3. (*Toto(1,2,999)*)

Edb must be declared as followed :

- *Predicatename(arg1,arg2,arg3).*
  OR
- *Predicatename().*

Idb rules must be declared as followed :**Head :- Subgoals**.
With **Head** like :

- *Predicatename(arg1,arg2,arg3).*
  OR
- *Predicatename().*

And **Subgoals** is a list of Predicate separated by **","** . You can add **not** before a predicate to negate it.
For example, declaration of an idb rule: *Toto(a) :- Beer(), Test("Ok"), Boat(a,b,999).*
Example of an idb rule with negation: *Toto(a) :- not Beer(), Test("Ok"), not Boat(a,b,999).*

To ensure that our syntax works well, we wrote many unitary tests. You can launch these tests with the following command: make tests (Tests require python testing package to work)
For more details about our syntax, you can check datalog.g4 files

4. Appendix

4.1. Subject example

4.1.1. Code

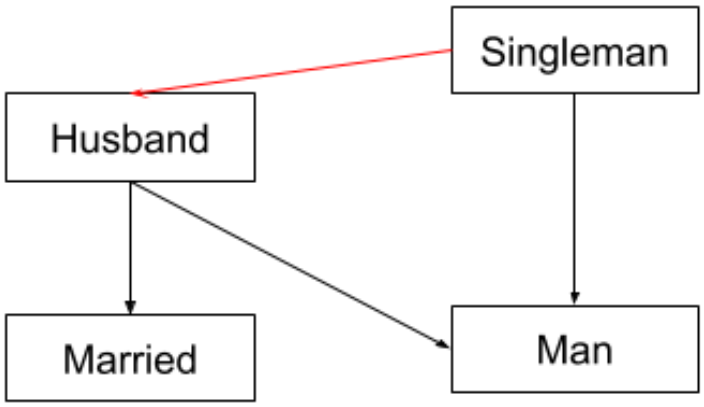
```
%edb
Person(pID,name,sex,married).

%idb
Man(x) :- Person(id,x,"M",m).
Singleman(x) :- Man(x), not Husband(x).
Married(x) :- Person(id,x,s,1).
Husband(x) :- Man(x), Married(x).
```

4.1.2. Result

```
P1 = { Man(x) :- Person(id,x,"M",m)
 Married(x) :- Person(id,x,s,1)
 Husband(x) :- Man(x), Married(x)}
P2 = { Singleman(x) :- Man(x), Husband(x)}
```

4.1.3. Graph



4.2. Example with many values

4.2.1. Code

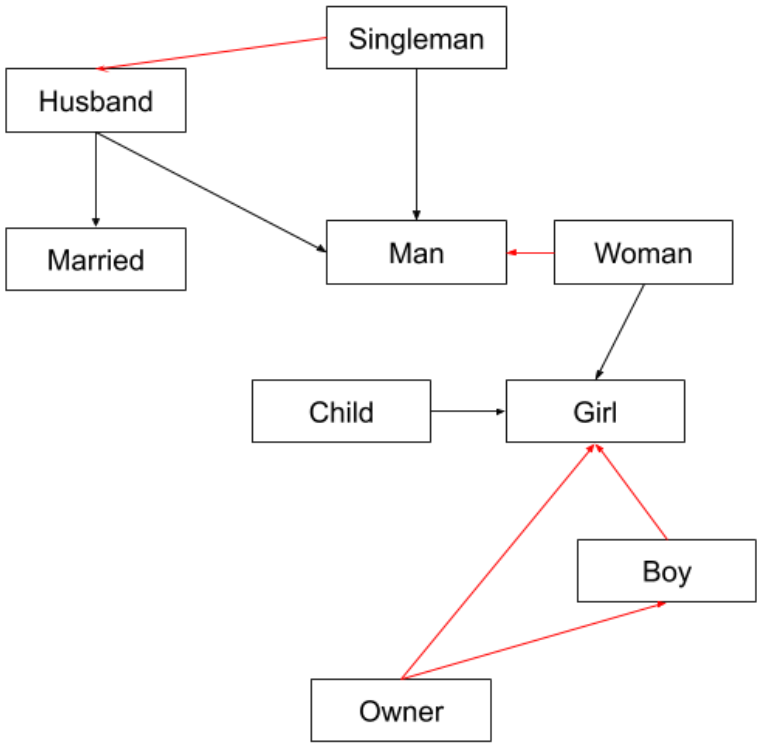
```
%edb
Person(pID,name,sex,married).
Home(hID, pId, address).
Child(pId, pIdH, pIDF).

%idb
Man(x) :- Person(id,x,"M",m).
Women(x) :- not Man(x).
Singleman(x) :- Man(x), not Husband(x).
Married(x) :- Person(id,x,s,1).
Husband(x) :- Man(x), Married(x).
Owner(hID, x):- Person(x,_,_,_), Home(hID, x,_,_), not Girl(x), not Boy(x).
Parent(x,y) :- Child(y, x, _).
Parent(x,y) :- Child(y, _, x).
Girl(x) :- Woman(x), Child(x,_, _).
Boy(x) :- not Girl(x).
```

4.2.2. Result

```
P1 = { Man(x) :- Person(id,x,"M",m)
 Married(x) :- Person(id,x,s,1)
 Husband(x) :- Man(x), Married(x)
 Parent(x,y) :- Child(y,x,_)
 Girl(x) :- Woman(x), Child(x,_,_)}
P2 = { Women(x) :- not Man(x)
 Singleman(x) :- Man(x), not Husband(x)
 Boy(x) :- not Girl(x)}
P3 = { Owner(hID,x) :- Person(x,_,_,_), Home(hID,x,_,_), not Girl(x),
not Boy(x)}
```

4.2.3. Graph



4.3. Example with several stratums

4.3.1. Code

```
%edb
Person(x).
Beer(x).
Vegetables(x).
Water(x).
Meat(x).

%idb
Livelonger(x):- Person(x), not Havebadhabits(x).
Havebadhabits(x):- not Eatvegetables(x), not Drinkwater(x), Person(x).
Eatmeat(x):- Love(x,y), Meat(y), not Eatvegetables(x).
Eatvegetables(x):- Love(x,y), Vegetables(y).
Drinkwater(x):- Love(x,y), Water(y), not Drinkbeer(x).
Drinkbeer(x):- Love(x,y), Beer(y).
```

4.3.2. Result

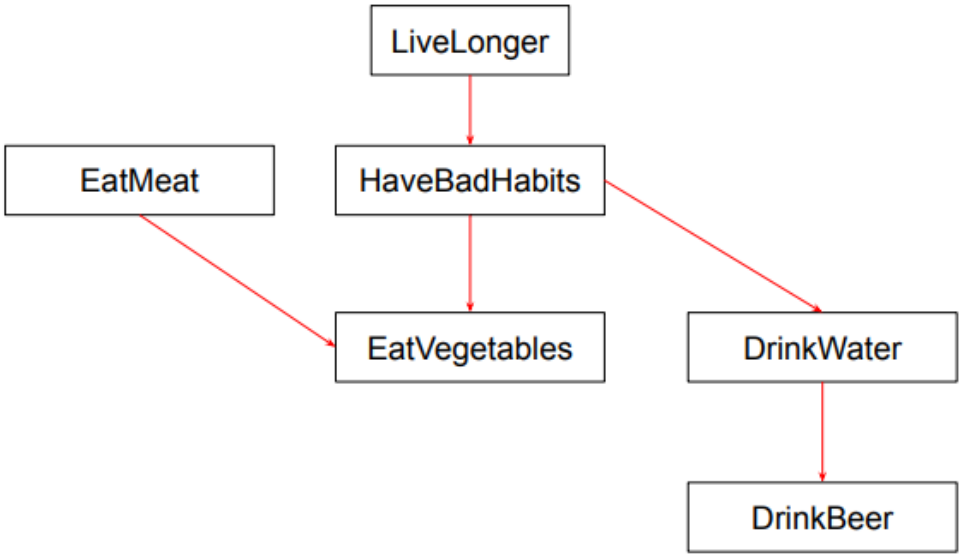
```
P1 = {
 Eatvegetables(x) :- Love(x,y), Vegetables(y)
Drinkbeer(x) :- Love(x,y), Beer(y)
}

P2 = {
 Eatmeat(x) :- Love(x,y), Meat(y), not Eatvegetables(x)
Drinkwater(x) :- Love(x,y), Water(y), not Drinkbeer(x)
}

P3 = { Havebadhabits(x) :- not Eatvegetables(x), not Drinkwater(x),
Person(x)}

P4 = { Livelonger(x) :- Person(x), not Havebadhabits(x)}
```

4.3.3. Graph



4.4. Example of two predicates with the same name but not the same arguments

4.4.1. Code

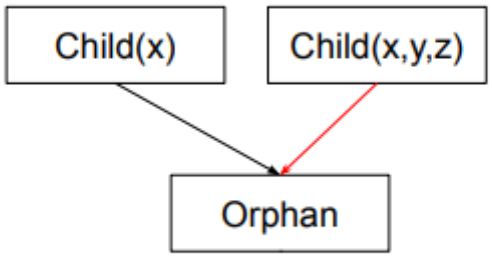
```
%edb
Person(x).
Alone(x).

%idb
Child(x):- Orphan(x).
Child(x,y,z):- not Orphan(x), Person(y), Person(z).
Orphan(x):- Alone(x).
```

4.4.2. Result

```
P1 = { Child(x) :- Orphan(x)
Orphan(x) :- Alone(x)}
P2 = { Child(x,y,z) :- not Orphan(x), Person(y), Person(z)}
```

4.4.3. Graph



4.5. Example of recursive predicate

4.5.1. Code

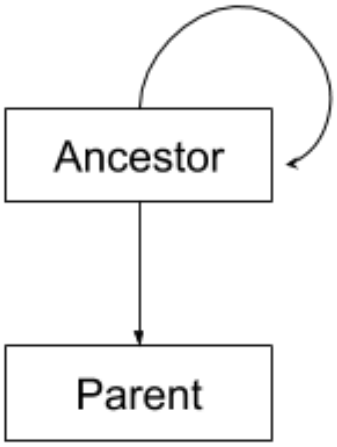
```
%edb
Person(pID,name,sex,married).
Child(pIDC, pIDP ).

%idb
Parent(x, y) :- Child(y,x).
Ancestor(x,y) :- Parent(x, y).
Ancestor(x,y) :- Parent(x,z), Ancestor(z,y).
```

4.5.2. Result

```
P1 = { Parent(x,y) :- Child(y,x)
 Ancestor(x,y) :- Parent(x,y)
 Ancestor(x,y) :- Parent(x,z), Ancestor(z,y)}
```

4.5.3. Graph



4.6. Example of recursive predicate with negation

4.6.1. Code

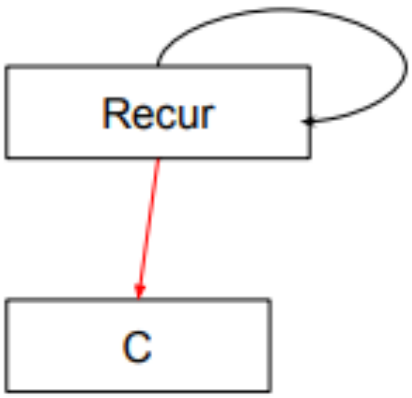
```
%edb
A(x).
B(y).

%idb
C(x,y) :- A(x), B(y).
Recur(x,y) :- not C(x, z), Recur(x,z).
```

4.6.2. Result

```
P1 = { C(x,y) :- A(x), B(y)}
P2 = { Recur(x,y) :- not C(x,z), Recur(x,z)}
```

4.6.3. Graph



4.7. Example with predicates with less dependency between them

4.7.1. Code

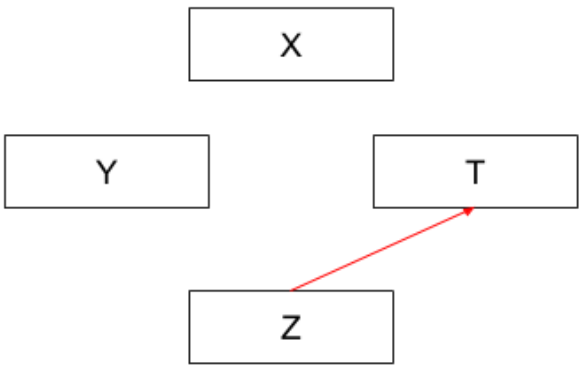
```
      %edb
A(a).
B().
C(t,u).


%idb
X() :- A("Hello").
Y(z,e) :- A(z), C(e,z).
Z() :- not B(), not T().
T() :- not A(5).
```

4.7.2. Result

```
P1 = { X() :- A("Hello")
Y(z,e) :- A(z), C(e,z)
T() :- not A(5)}
P2 = { Z() :- not B(), not T()}
```

4.7.3. Graph



4.8. Example with circular dependency

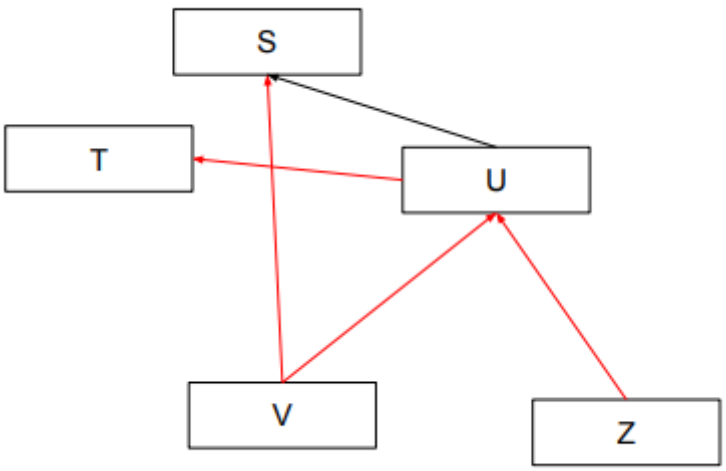
4.8.1. Code

```
%idb
S() :- R(x), Rr(x).
T(x,y) :- not R(x), not Rr(x).
T(x,y) :- not R(x), not Rr(x), T(x,y).
T(x,y) :- not R(x), T(y,x).
U(x) :- R(x), S(), not T(1,2).
Z(x) :-  not U(x).
V(x) :- not S(), not Z(x).
```

4.8.2. Result

```
P1 = { S() :- R(x), Rr(x)
T(x,y) :- not R(x), not Rr(x)
T(x,y) :- not R(x), not Rr(x), T(x,y)
T(x,y) :- not R(x), T(y,x)
}
P2 = { U(x) :- R(x), S(), not T(1,2)}
P3 = { Z(x) :- not U(x)}
P4 = { V(x) :- not S(), not Z(x)}
```

4.8.3. Graph



4.9. Complexe example with recursion and predicate with the same name but not the same arguments.

4.9.1. Input

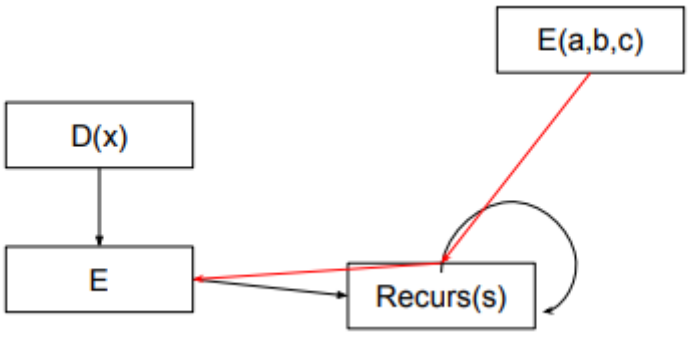
```
%edb
A(x).
Toto(z).


%idb
D(x) :- A(x), E().
E() :- A(x), Recurs(s).
Recurs(x) :- A(x), not E().
Recurs(x) :- Toto(x), Recurs(1).
E(a,b,c) :- Toto(a), not Recurs(c).
```

4.9.2. Result

```
P1 = {
D(x) :- A(x), E()
E() :- A(x), Recurs(s)
}
P2 = {
Recurs(x) :- A(x), not E()
Recurs(x) :- Toto(x), Recurs(1)
}
P3 = {
E(a,b,c) :- Toto(a), not Recurs(c)
}
```

4.9.3. Graph



4.10. Example with all in P1

4.10.1. Code

```
%idb
S() :- R(x), Rr(x).
T(x,y) :- R(x), Rr(x).
T(x,y) :- R(x), Rr(x), T(x,y).
T(x,y) :- R(x), T(y,x).
U(x) :- R(x), S(), T(1,2).
Z(x) :-  U(x).
V(x) :- S(), Z(x).
```

4.10.2. Result

```
P1 = { S() :- R(x), Rr(x)
 T(x,y) :- R(x), Rr(x)
 T(x,y) :- R(x), Rr(x), T(x,y)
 T(x,y) :- R(x), T(y,x)
 U(x) :- R(x), S(), T(1,2)
 Z(x) :- U(x)
 V(x) :- S(), Z(x)}
```

4.10.3. Graph