Datalog

Datalog

- More detailed theoretical foundations at Databaseslectures
- A program in a Datalog is a set of rules (implications), e.g.
  - zlozene_cislo(Z) ← krat(X, Y, Z), int(X), int(Y), not X = 1, not Y = 1.
  - prvocislo(Z) ← int(Z), not Z = 1, not zlozene_cislo(Z).
- Syntax:

```
<hlava>: <atom>
<hlava> :- <telo>
<telo>: <atom> | \+ <atom> | <telo>, <atom>
```

We will use Prolog to evaluate Datalog queries, so we will use Prolog syntax (which is a superset of Datalog) to write them:

- `\+` is negation, `:-` is "implication"

Datalog

- Example of a datalog rule:
  - res(N,J) :- emp(_,N,J,_,_,S,_), S>=2000.
- On the left side, only one positive atom at a time
- Variables start with a capital letter
- Constants in lowercase
- Each variable is listed in at least one positive EDB context in the body of the rule
- _ means anonymous variable
- The ***is*** operator is used to evaluate arithmetic expressions:
  - E.g. X is 2+3, not X = 2+3
  - (the = symbol would be interpreted as the unification of the therms and no arithmetic operation will occur).

Working with datalog: SWI-Prolog

- Three options:
  - on servers *cvika*, login using ssh on cvika.dcs.fmph.uniba.sk

    (username/password from AISe)
  - using SWI-Prolog on your local computer
  - online at https://swish.swi-prolog.org/

- We recommend opening 3 windows
  - In one, you edit a file with queries, eg. **vim queries_emp.pl**
  - In the second window, you are running the Prolog environment: **swipl -s queries_emp.pl**
  - in the third window you have a database (list of facts)

Working with datalog

- After you write a query to a file, you need to save it to disk
  (vim: ESC, ":w", ENTER).
- then compile the new version: **make.** (even with that dot)
  - Be sure to check if the compiler reports errors and fix them if necessary
- Calculation of queries:
  ```
  ?- q(job(J)).
  ```
- the predicate "q(_)" is used to nicely format the output and eliminate apparent duplicates (Prolog does full backtracking and can find a specific value in multiple branches)

Datalog and negation

- List touples [D, J], where job J is not in the department D:

```
jobDept(J, D) :- emp(_,_,J,_,_,_,D).
missing(D, J) :-
    emp(_,_,J,_,_,_,_),
    emp(_,_,_,_,_,_,D),
    \+ jobDept(J,D).
```

- Why we cannot write the following program?

```
missing(D, J) :-
        emp(_,_,J,_,_,_,_),
    emp(_,_,_,_,_,_,D),
    \+ emp(_,_,J,_,_,_,D).
```

Datalog and general quantifier

- It is necessary to rewrite the general quantifier as a negation of the existential, so in the auxiliary rule we describe the counterexample and by negating the auxiliary predicate we say that the counterexample does not exist

- departments in which each type of work is represented:

```
hasAllJobs(D) :- dept(D,_,_,_),\+ missingJob(D).
missingJob(D) :- emp(_,_,J,_,_,_,_), emp(_,_,_,_,_,_,D),
                 \+ jobDept(J,D).
jobDept(J, D) :- emp(_,_,J,_,_,_,D).
```