DATALOG: Logic-based Information Management

**Datalog₀**: IM based on subset of FO logic

*Example*: want to manage information about family relationships

- **Tell** facts like “parents of chuck are liz and phil; chuck is male;”
- **Ask** questions like “Is chuck male? Is phil a father? Who is chuck’s mother? Who are all the males?”

Use notation of FOL for both languages.

- **predicates/relations**: male, parents
- **constants**: ed, vicky, ... *(must start with lowercase)*
- **ground atomic formula**: \(<\text{pred}>\)(<\text{const}>,...).
- \(\mathcal{L}_{\text{tell}} = \) ground atomic formulas

---

We can (sort of) use the Prolog programming language as a Datalog IM implementation.

```
parents(chuck,liz,phil).
parents(ann,liz,phil).
parents(andy,di,chuck).
% parents(child,mother,father)
```

\(\mathcal{L}_{\text{question}} = \mathcal{L}_{\text{tell}}\)

\(\mathcal{L}_{\text{answer}} = \{\text{yes,}\text{no}\}\)

(In lectures I will be using an on-line PROLOG interpreter: http://swish.swi-prolog.org)
Datalog: queries with variables

- **variables**: Y, A, Who (by convention, start with caps)
- **argument**: constant or variable
- **atomic formula**: \(<\text{predicate}>((<\text{argument}>),\ldots)\).

\(L_{\text{question}} = \text{atomic formulas (may have variable)}\)
\(L_{\text{answer}} = \{\text{yes},\text{no}\} \text{ or variable substitutions}\)

\(\text{likes(eve,pie). person(tom).} \)
\(\text{likes(al,eve). food(pie).} \)
\(\text{likes(eve,tom). food(apple).} \)
\(\text{likes(eve,eve).} \)

(Aside: Datalog is more “elegant/brief” than SQL)

Datalog:

- \(\text{likes(eve,pie). person(tom).} \)
- \(\text{likes(al,eve). food(pie).} \)
- \(\text{likes(eve,tom). food(apple).} \)
- \(\text{likes(eve,eve).} \)

?\(-\text{likes(al,Whom)}. \)

Corresponding SQL:

CREATE TABLE likes(who VARCHAR; whom VARCHAR;)
PRIMARY KEY(who,whom);
INSERT INTO likes VALUES ('eve','pie');
...

SELECT t.whom FROM likes t
WHERE t.who='al';

Datalog: conjunctive queries

\(\text{likes(eve,pie). person(tom).} \)
\(\text{likes(al,eve). food(pie).} \)
\(\text{likes(eve,tom). food(apple).} \)
\(\text{likes(eve,eve).} \)

?\(-\text{likes(eve,W)}, \text{person(W)}. \)
\(W=tom\)
?\(-\text{likes(eve,V)}, \text{likes(al,V)}. \)
\(V=eve\)
?\(-\text{likes(eve,W)}, \text{likes(al,V)}, V=W. \)
\(V=eve\)
?\(-\text{likes(eve,W)}, \text{person(W)}, \text{food(V)}. \)
\(W=tom,V=pie; W=tom,V=apple\)
?\(-\text{likes(eve,W)}, \text{likes(W,V)}. \)
\(W=eve,V=tom; W=eve,V=eve\)
### Datalog RULES

**a) Rules as shorthand for some queries**

```datalog
likes(eve,pie). person(tom).
likes(al,eve). food(pie).
likes(eve,tom). food(apple).
likes(eve,eve).
```

**q1:** `- likes(eve,V), person(V).`

?-q1.

`yes`

"Is there someone whom Eve likes?" (hides uninteresting vars)

**q2:** `- likes(W,F), food(F).`

?-q2(H).

`H=eve`

"Who likes some food?" (hides some variables not of interest -- the food F liked by that person)

**b) Rules for error checking**

```datalog
likes(eve,pie). person(tom).
likes(al,eve). food(pie).
likes(eve,tom). food(apple).
likes(eve,eve).
```

**error:** `- likes(X,_,food(X)).
error :- food(Y),person(Y).`

?-error.

These are not queries but rules added to the "program!"

Note possibility of multiple rules defining the same idea.
Return YES if any say YES. Return NO if all fail.

**c) Rules for defining new predicates**

```datalog
male(phil). female(liz).
male(chuck). female(di).
parents(chuck,liz,phil).
parents(ann,liz,phil).
parents(andy,di,chuck).
%% parents(child,mother,father)
sibling(X,Y):-parents(X,M,F),parents(Y,M,F),NOT(X=Y).
```

A rule has
- **head** -- atom (sometimes called goal)
- **body** -- conjunctive query (atoms sometimes called subgoals)
- can be interpreted naturally as "conclude head if body", with variables not appearing in the head quantified as "there exists" at the left end of the body

"sibling(X,Y) IF exist M,F such that parents(X,M,F) and parents(Y,M,F) and NOT (X=Y)"

**How to think when writing rules**

Define **uncleOf(U,N):** "U(nce) is the uncleOf N(ephew/iece)".
- U must be a male **male(U)**
- and there must be some parent P of N (either **parents(N,P,_)** or **parents(N,_,P)**
- so that U is a sibling of that parent **sibling(U,P)**
- Putting these together, we need two rules:

  **uncleOf(U,N) :- male(U), parents(N,P,_), sibling(U,P).**
  **uncleOf(U,N) :- male(U), parents(N,_,P), sibling(U,P).**

The order of the atoms on the right hand side and the order of the rules does not matter in logic & Datalog. (It matters in Prolog ☝️)
Result

male(phi). female(liz).
male(chuck). female(di).
parents(chuck,liz,phi).
parents(ann,liz,phi).
parents(andy,di,chuck).
%% parents(child,mother,father)
sibling(X,Y):-parents(X,M,F),parents(Y,M,F), NOT(X=Y).
uncleOf(U,N):-male(U), parents(N,P,_), sibling(U,P).
uncleOf(U,N):-male(U), parents(N,_,P), sibling(U,P).

Datalog\textsubscript{0.7}: Datalog\textsubscript{0.5} + rules

\[ \mathcal{L}_{tell1} = \text{ground atomic formulas} \]
\[ \mathcal{L}_{tell2} = \text{rules} \]
\[ \mathcal{L}_{question}, \mathcal{L}_{answer} \text{ as before} \]

likes(eve,pie). person(tom).
likes(al,eve). food(pie).
likes(eve,tom). food(apple).
likes(eve,eve).
likeable(V):-likes(W,V),person(V).
wellLiked(X):-likes(W,X),likes(V,X),W\neq V.

• In Datalog, those predicates specified by listing the arguments at which they are true are called \textit{input/extensional/edb/base} predicates
• In Datalog, predicates described by rules are called \textit{output/intensional/idb/view} predicates.
• The above two sets of predicates must be disjoint in Datalog

“Safety”: a problem in Datalog\textsubscript{0.7}

\begin{align*}
\text{age(tom,1).} \\
\text{age(eve,70).} \\
\text{old_ages1(A)} & : \text{age}(P,A), A > 60. \\
\text{old_ages2(A)} & : A > 60. \\
? \text{- old_ages1(A)} & \\
70 \\
? \text{- old_ages2(A)} & \\
\text{some kind of unexpected answer (NOT an infinite enumeration of integers)}
\end{align*}

Safe query/rule: every variable in a rule head must appear in a (non-negated) predicate other than arithmetic one

Datalog\textsubscript{Neg} = Datalog\textsubscript{0.7} + Negation

likes(eve,pie). person(tom).
likes(al,eve). food(pie).
likes(eve,tom). food(apple).
likes(eve,eve).

?- \text{likes}(eve,W), \text{NOT likes}(al,W).
W=pie ; W=tom
?- \text{NOT likes}(_,Y).
\text{UNSAFE QUERY!}

\begin{align*}
\text{thing}(X) & : \text{likes}(X,\_). \\
\text{thing}(X) & : \text{likes}(\_,X). \\
\text{thing}(X) & : \text{food}(X). \\
\text{thing}(X) & : \text{person}(X). \\
\text{disliked}(Y) & : \text{thing}(Y), \text{NOT likes}(\_,Y).
\end{align*}

Now it is safe
Beware of Negation Meaning

\[ \text{likes(eve, pie)}. \quad \text{person(tom)}. \quad \text{person(eve)}. \quad \text{person(al)}. \]
\[ \text{likes(al, eve)}. \quad \text{food(pie)}. \]
\[ \text{likes(eve, tom)}. \quad \text{food(apple)}. \]
\[ \text{likes(eve, eve)}. \]
\[ \text{liked(P)} \iff \text{person(P)}, \text{likes(L,P)} \]
\[ \text{hated(P)} \iff \text{person(P)}, \overline{\text{liked(P)}}. \]
\[ \text{who(P)} \iff \text{person(P)}, \overline{\text{likes(L,P)}}. \]

?liked(X) \iff X = \text{tom}, \text{eve}
?hated(X) \iff X = \text{al}
?who(X) \iff X = \text{tom}, \text{eve}, \text{al}

Datalog\_REC = \text{Datalog}_{0.7} + \text{recursive rules}

\( \text{male(phil)}. \quad \text{female(liz)}. \)
\( \text{male(chuck)}. \quad \text{female(di)}. \)
\( \text{parents(chuck,liz,phil)}. \)
\( \text{parents(ann,liz,phil)}. \)
\( \text{parents(andy,di,chuck)}. \)
\( \% \% \text{parents(child,mother,father)} \)

\( \text{parentOf(P,C)} :\iff \text{parents(C,P,)} \).
\( \text{parentOf(P,C)} :\iff \text{parents(C,)}, \text{P} \).
\( \text{grandParentOf(GP,GC)} :\iff \text{parentOf(GP,P)}, \text{parentOf(P,GC)}. \)
\( \text{ancestorOf(X,Y)} :\iff \text{parentof(X,Y)}. \)
\( \text{ancestorOf(X,Y)} :\iff \text{parentOf}(X,Z), \text{ancestorOf}(Z,Y). \)

So, who finds persons who do not like SOME food

\begin{align*}
?\text{liked}(X) & \iff X = \text{tom, eve} \\
?\text{hated}(X) & \iff X = \text{al} \\
?\text{who}(X) & \iff X = \text{tom, eve, al}
\end{align*}

Datalog\_REC = \text{Datalog}_{0.7} + \text{recursive rules}

General application of recursion: computing transitive closure of a relationship, in anything that can be represented as a graph

\( \text{g(1,2)}. \quad \text{g(2,3)}. \quad \text{g(3,2)}. \)

\( \text{tr}(X,Y) :\iff \text{g}(X,Y). \)
\( \text{tr}(X,Y) :\iff \text{g}(X,Z), \text{tr}(Z,Y). \)

Trace \( \text{tr}(X,Y) \) to see why the answers are

1, 2
2, 3
3, 2
1, 3
2, 2
3, 3

Datalog = Datalog\_REC + Datalog\_Neg

\( \text{male(phil)}. \quad \text{female(liz)}. \)
\( \text{male(chuck)}. \quad \text{female(di)}. \)
\( \text{parents(chuck,liz,phil)}. \)
\( \text{parents(ann,liz,phil)}. \)
\( \text{parents(andy,di,chuck)}. \)
\( \% \% \text{parents(child,mother,father)} \)

\( \text{parentOf(P,C)} :\iff \text{parents(C,P,)} \).
\( \text{parentOf(P,C)} :\iff \text{parents(C,)}, \text{P} \).
\( \text{grandParentOf(GP,GC)} :\iff \text{parentOf(GP,P)}, \text{parentOf(P,GC)}. \)
\( \text{ancestorOf(X,Y)} :\iff \text{parentof(X,Y)}. \)
\( \text{ancestorOf(X,Y)} :\iff \text{parentOf}(X,Z), \text{ancestorOf}(Z,Y). \)

(note “recursion” on \text{ancestorOf} in last rule)
Trace calls to \text{parentOf}, \text{grandParentOf}, \text{ancestorOf}
Recitation: Example domain1