Deductive Databases

Datalog & Prolog

Motivation

- SQL-92 cannot express some queries: Possible in SQL-99 but hard!
  - Are we running low on any parts needed to build a Jaguar sport car?
  - What is the total component and assembly cost to build a Jaguar sport car at today's part prices?
- Can we extend the query language to cover such queries?
  - Yes, by adding recursion.
- SQL queries can be read as:
  - If some tuples exist in the From tables that satisfy the Where conditions, then the Select tuple is in the answer."
- Datalog is a query language that has the same if-then structure
  - We will use Prolog, as we will follow a Prolog-like syntax

DDBs

- DDBs is an area at the intersection of databases, AI, and logic
- A DDB is a database system including capabilities to define deductive rules, via which we can deduce or infer additional information from the facts stored in a database
- Theoretical foundation for DDBs is mathematical logic
- Other types of systems (e.g. expert database systems or knowledge-based systems) also incorporate reasoning and inference capabilities
  - Commonality: both use safe foundations
  - Difference: DDBs makes use of data in secondary storage, expert database systems assume that the data needed resides in main memory

DDBs - Properties

- In DDBs se specify rules through a declarative language
  - Focus is on WHAT and not HOW
- An inference engine or deduction mechanism, in the DDB can deduce new facts from the database by interpreting these rules
- The model used by DDBs is related to:
  - Relational Data Model (Domain Relational calculus)
  - Logic Programming (Prolog)
- A variation of Prolog, called Datalog, is used to define rules declaratively
  - The existing set of “relations” is treated as a set of literals in the language
  - Datalog syntax is very similar to Prolog
  - Datalog semantics **is NOT** equal to Prolog semantics

Main Idea of Deductive Database Systems (DDBs)

Specify Facts, Rules and a framework to **infer** (deduce) new information, based on those facts and rules

- The DDB system will provide the framework*
- The programmer will provide facts and rules
- DDBs are very common in the fields of Expert Systems and Knowledge-Based Systems

* Prolog has a built-in backward chaining inference engine

Inference Mechanism*

- Typically, two approaches for evaluating logical programs:
  - Bottom-up (also called forward chaining and bottom-up resolution)
    - The inference engine starts with the facts and applies the rules to generate new facts. That is, the inference moves forward from the facts toward the goal.
    - As facts are generated, they are checked against the query predicate goal for a match.
  - Top-down (also called backward chaining and top-down resolution)
    - The inference engine starts with the query goal and attempts to find matches to the variables that lead to valid facts in the database. That is, the inference moves backward from the intended goal to determine facts that would satisfy the goal.
    - During the process, the rules are used to generate subgoals. The matching of these subgoals will lead to the match of the intended goal.

Using a Rule to Deduce New (Tuples) Information*

- Each rule is a template
  - by assigning constants to the variables in such a way that each body “literal” is a tuple in the corresponding relation, we identify a tuple that must be in the head relation.
- This is called an inference using the rule
- Given a set of tuples, we apply the rule by making all possible inferences with these tuples in the body

* Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke.

Deductive Systems (circa 2012)

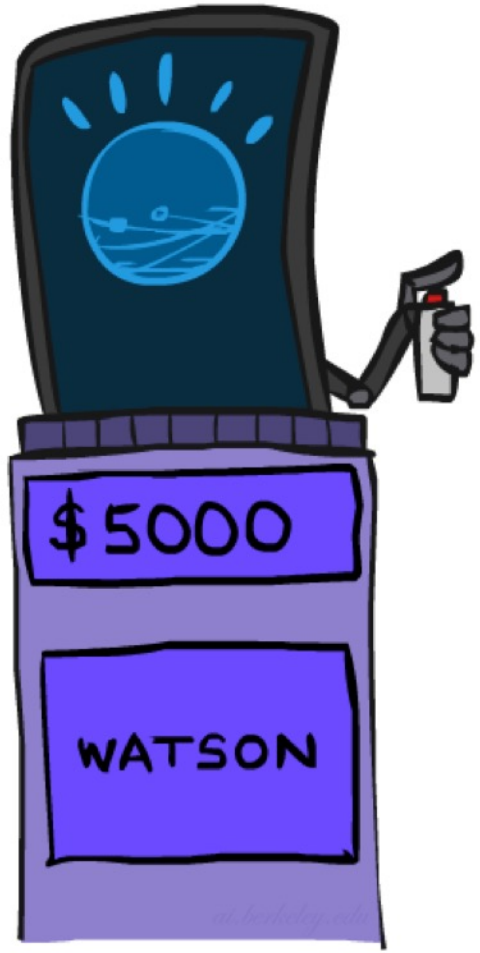
Classic academic deductive systems

- LDL++ (UCLA)
- CORAL (Univ. of Wisconsin)
- NAIL! (Stanford University)

Ongoing developments

- Recent commercial deductive systems
  - BOOM/Dedalus (Berkeley)
  - DLV (Italy, University of Calabria)
  - LogicBlox (USA)
  - Intellidimension (USA)
  - Semmle (UK)
- Recent academic deductive systems
  - 4QL (Warsaw University)
  - bddbddb (Stanford University)
  - ConceptBase (Passau, Aachen, Tilburg Universities, since 1987)
  - XSB (Stony Brook University, Universidade Nova de Lisboa, XSB, Inc., Katholieke Universiteit Leuven, and Uppsala Universitet)
  - DES (Complutense University)

IBM-Watson



"a camel is a horse designed by"

Wiktionary
a multilingual free encyclopedia
['wɪkʃənrɪ] *n.*, a wiki-based Open Content dictionary

Main Page
Community portal
Preferences
Requested entries
Recent changes
Random entry
Help
Donations
Contact us
Toolbox
What links here
Related changes
Upload file
Special pages
Printable version
Permanent link
In other languages
Français
Русский

Log in / create account
Entry | Discussion | Read | Edit | History | Search

a camel is a horse designed by a committee

Contents [hide]
1 English
1.1 Alternative forms

The Phrase Finder

> Discussion Forum

Google™ Custom Search | Search

A camel is a horse designed by committee

Posted by Ruben P. Mendez on April 16, 2004

Does anyone know the origin of this maxim? I heard it way back at the United Nations, which is chockfull of committees. It may have originated there, but I'd like an authoritative explanation. Thanks

- Re: A camel is a horse designed by committee **SR** *16/April/04*
  - Re: A camel is a horse designed by committee **Henry** *18/April/04*

IBM Watson - UIMA Prolog Interface

LAT: Lexical Answer Types

CAS: Common Analysis Structure

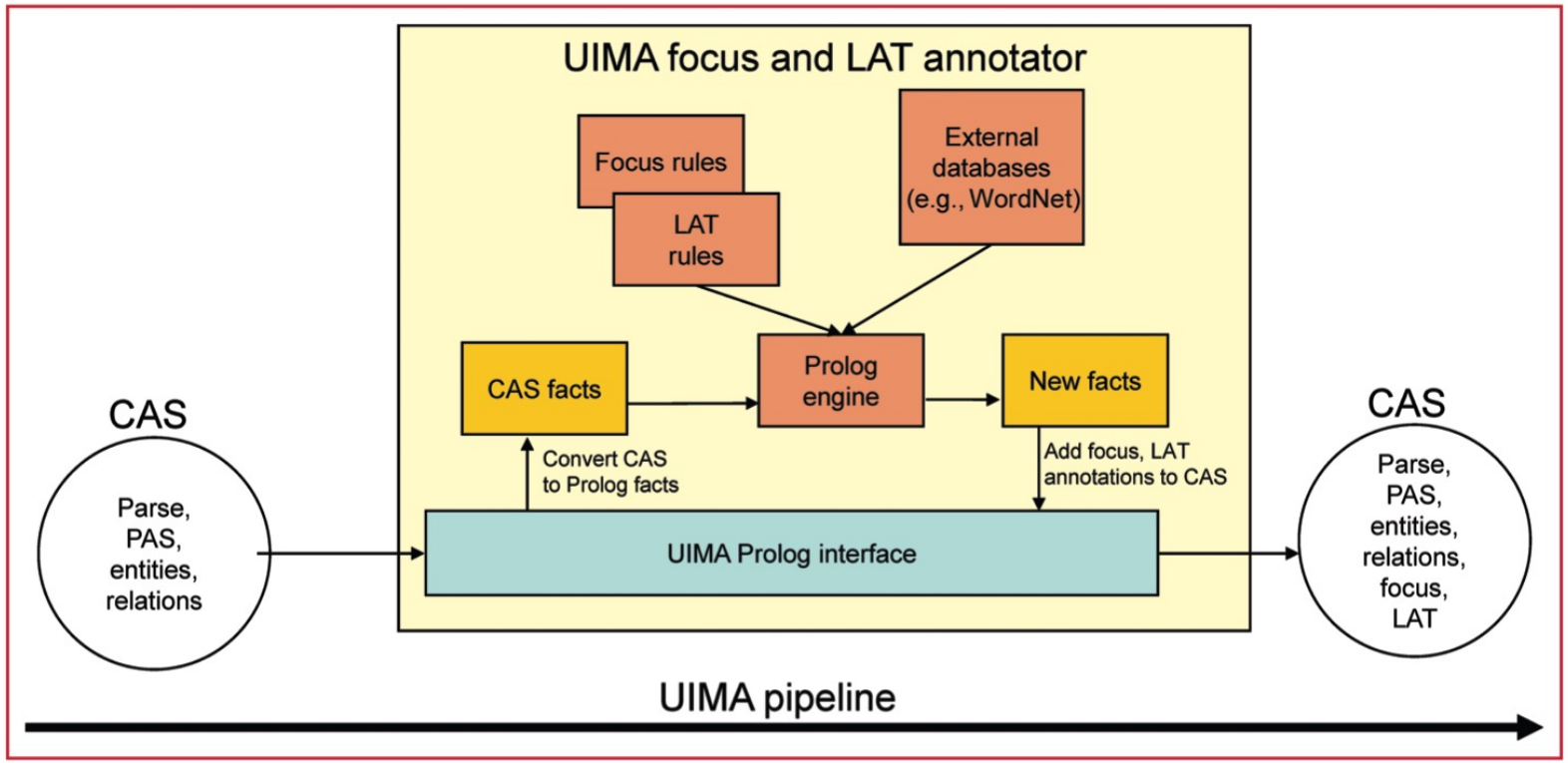


Figure 1

UIMA Prolog interface specialized for focus and LAT detection. (Figure used with permission from P. Fodor, A. Lally, and D. Ferrucci, "The Prolog Interface to the Unstructured Information Management Architecture," Computing Research Repository, 2008; available: http://arxiv.org/abs/0809.0680v1.)

WordNet: WordNet® is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept.

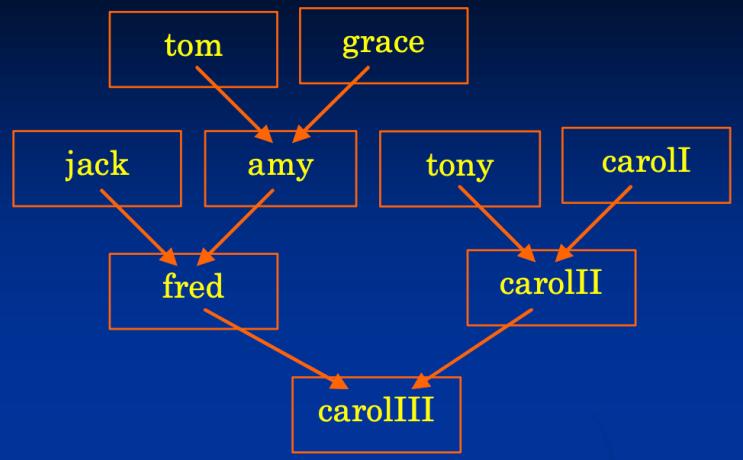
Prolog – Basic Syntax*

- Predicates should have unique names, that convey meaning, and a fixed number of arguments
- Constants are expressed in **lower** case and variables must start with a capital letter
- If the arguments are all constant values, then the predicate states that a certain fact is TRUE
- If the arguments are variables, then the predicate is considered as a query or as a part of a rule or constraint
- Using Prolog syntax:
  - Constants values in a predicate are either numeric or strings (lower case only)
  - Variable names start with an uppercase letter

* Kotsireas, Ilias. DDBs Notes. Wilfrid Laurier University, Ontario, CA.

Datalog Example (Complutense– Madrid)

Datalog Example



- Facts:

```
father(tom,amy).
father(jack,fred).
father(tony,carolII).
father(fred,carolIII).

mother(graceI,amy).
mother(amy,fred).
mother(carolI,carolII).
mother(carolII,carolIII).
```

- Rules:

```
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

- Query:

```
parent(X,Y)
```

- Minimal model for **parent**:

```
{
   (tom,amy), (grace,amy), (jack,fred),
   (amy,fred), …
}
```

```
% FACTS
father(tom, amy).
father(jack, fred).
father(tony,carolII).
father(fred,carolIII).

mother(graceI, amy).
mother(amy, fred).
mother(carolI, carolII).
mother(carolII, carolIII).

% RULES
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

But we will use Prolog (Programming in Logic). Datalog is a subset of Prolog

Datalog Example (Complutense– Madrid)

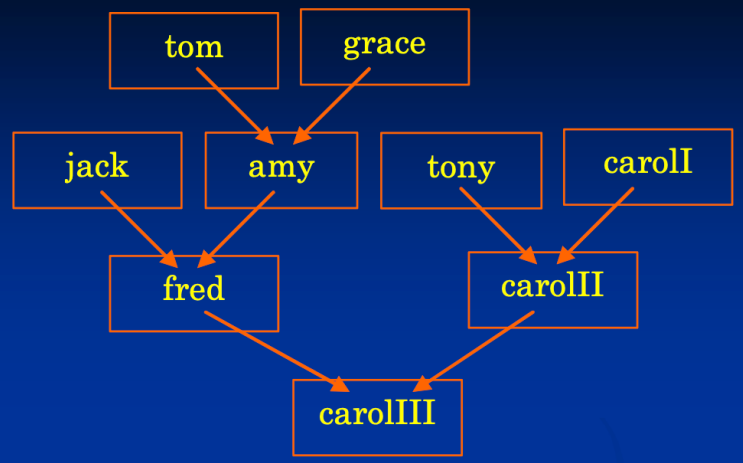
Recursion

```
father(tom,amy).
father(jack,fred).
father(tony,carolII).
father(fred,carolIII).

mother(graceI,amy).
mother(amy,fred).
mother(carolI,carolII).
mother(carolII,carolIII).

parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).

ancestor(X,Y) :-
   parent(X,Y).
ancestor(X,Y) :-
   parent(X,Z),
   ancestor(Z,Y).
```



DES implements a fixpoint semantics for recursive Datalog, finding the least fixpoint as answer

```
ancestor(tom,X)

{
  ancestor(tom,amy),
  ancestor(tom,carolIII),
  ancestor(tom,fred)
}
```

```
% FACTS
father(tom, amy).
father(jack, fred).
father(tony,carolII).
father(fred,carolIII).

mother(graceI, amy).
mother(amy, fred).
mother(carolI, carolII).
mother(carolII, carolIII).

% RULES
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
% Adding recursion
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z),
                 ancestor(Z,Y).
```

What is Datalog*?

- Datalog is a query language for deductive databases.
- The language queries a database by creating rules and executing known facts on those rules.
- The rules create more facts and continue to add facts to the database until no more facts can be derived.
- The rules are based on the information in the database.
  - If a graph is stored in the database, the user can store edges and nodes as known facts and write rules that find the paths between individual nodes based on the edges in the graph.
- Datalog has a variety of applications including data integration, declarative networking, program analysis, information extraction, network monitoring, security, and cloud computing

Capooci, Corey. Performance Comparison of Three Datalog Engines. The Pennsylvania State University (2018)

What is Datalog*?

- A Datalog engine is software that implements the concepts of Datalog.
- Private sector and universities develop Datalog engines.
- Some versions of Datalog are open-source and others, such as LogicBlox, are for-profit.
- Nevertheless, all of the Datalog engines allow the ability to model problems and query results with a Datalog language.

Capooci, Corey. Performance Comparison of Three Datalog Engines. The Pennsylvania State University (2018)

What is Datalog*?

- The basis of the Datalog system is the predicate, or relation.
- In Datalog, there are two kinds of predicates:
  - Extensional database predicates (EDB)
    - Are the known facts in the program.
    - The EDB do not change as the Datalog program is executed
  - Intensional database predicates (IDB)
    - The rules defined by the programmer create the IDB.
    - As the program is running, the rules generate IDB, which could generate more IDB and so on.
    - The Datalog program will continue until no more IDB are generated.

Capooci, Corey. Performance Comparison of Three Datalog Engines. The Pennsylvania State University (2018)

What is Datalog*?

- To use Datalog, the programmer creates the rules and the facts into files.
- The syntax for defining a rule is <head> <operator> <body>.
  - The head and the body are a list of predicates.
  - If all of the predicates in the body exist, then the rule generates the predicates in the head as IDB.
  - The operator varies depending on the engine, but for this class we will use only “:-“
  - The operator delineates the head from the body and dictates that the predicates in the body imply the predicates in the head.

Capooci, Corey. Performance Comparison of Three Datalog Engines. The Pennsylvania State University (2018)

What is Datalog*?

- A predicate must be IDB or EDB, not both.
  - An IDB predicate can appear in the body or head of a rule
  - EDB only in the body

Capooci, Corey. Performance Comparison of Three Datalog Engines. The Pennsylvania State University (2018)

Simple Datalog program with three variables: X, Y and Z

```
% First rule says that if there is an edge from node X to
% node Y then there is a path from node X to node Y.
path(X,Y) :- edge(X,Y).
path(X,Z) :- path(X,Y), edge(Y, Z).          % the comma is an "AND"


% facts stating edges
edge(1,2).
edge(3,4).
edge(4,5).
```

Capooci, Corey. Performance Comparison of Three Datalog Engines. The Pennsylvania State University (2018)

Expressive power of Datalog*

- Non-recursive Datalog = Relational Algebra.
- Datalog simulates SQL select-from-where without aggregation and grouping.
- Recursive Datalog expresses queries that cannot be expressed in SQL.
- Neither Datalog nor SQL have full expressive power (Turing completeness)

Nestorov, Svetlozar. Introduction to Databases (University of Chicago, USA).

Datalog: Negation and Recursion*

- Negation wrapped inside a recursion makes no sense.
- Even when negation and recursion are separated, there can be ambiguity about what the rules mean, and one meaning must be selected.
- Stratified Negation is the answer
  - It is an additional restraint on recursive rules (like safety) that solves both problems:

    1. It rules out negation wrapped in recursion.

    2. When negation is separate from recursion, it yields the intuitively correct meaning of rules (the stratified model).

Nestorov, Svetlozar. Introduction to Databases (University of Chicago, USA).

Prolog, Datalog and Relational Algebra(RA)

Title	Year	Length	Type
Harry Porter	2001	180	Color
Gone with the wind	1938	125	Black and White
Snow White	1950	90	Color

Prolog, Datalog & RA

```
colorMovie(M) :- movie(M,X,Y,Z), Z = "Color".

longMovie(M):- movie(M,X,Y,Z), Y>120.

oldMovie(M):- movie(M,X,Y,Z), X<1940.

movieWithSequel(M):- movie(M,X1,Y1,Z1),
                     movie(M,X2,Y2,Z2),
                     X1 <> X2.
```

Prolog, Datalog and RA

- Projection

  colorMovie(M) :- movie(M,_,_,Z), Z = *“Color”.*
- Selection

  oldMovie(M,X,Y,Z):- movie(M,X,Y,Z),X<1940.
- Join (two relations p(A,B), q(B,C))

  r(X,Y,Z):- p(X,Y), q(Y,Z)
- Union (two relations p(A,B), q(A,B))

  r(X,Y):- p(X,Y).

  r(X,Y):- q(X,Y).
- Set difference (two relations p(A,B), q(A,B))

  r(X,Y):- p(X,Y), not q(X,Y).
- Cartesian product (two relations p(A,B), q(C,D))

  c(X,Y,Z,W):- p(X,Y), q(Z,W).