

DATABASE THEORY: WEEK #X

Reminder: **CS Department Awards Ceremony**

4 June, 4:00-6:00 PM 400ABC Tangeman University Cntr.

All CS students (graduate & undergraduate) are invited.

Agenda:

Food, pop, coffee, etc.:	4:00 PM.
Awards presentation:	4:45 PM
Talk:	≈ 4:55-5:30 PM

Speaker: Dave Conard, Collaboration Services Leader for all of GE since 2006. Before that he was Chief Technology Officer for IT Infrastructure standards and projects in GE. He is a 1983 graduate of UC's college of Engineering (Industrial Engg).

RSVP: to Teresa Hamad (teresa.hamad@uc.edu) to help us plan how much food to get (**¡you don't want us to mess up on that!**) and to get more details.

Announcement from ECE-GSA to ECE/CS students:

The GSA is pleased to announce the third seminar in the programming seminar series [covering] basic concepts of Perl.

If you are interested in attending, please [register at]

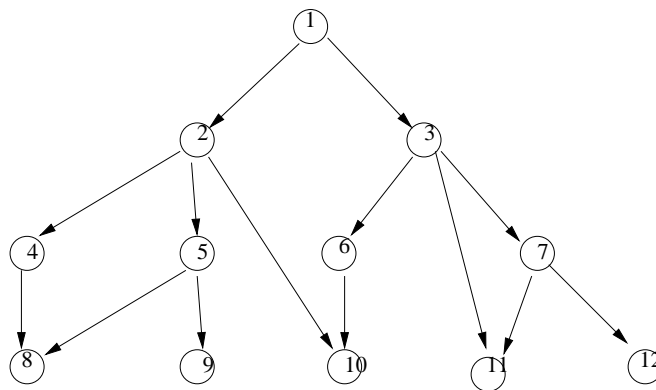
http://www.surveymonkey.com/s.aspx?sm=p8ycn9UC45Z6G7dn6I6aiw_3d_3d

Baldwin 860D Thursday June 5, 2008 4.00 pm - 5.30 pm

Deductive Databases:

- Add methods for making — and accumulating — logical inferences ...
- **Paradigm Examples:**
 1. Given a database relation *parentOf*, infer a relation *ancestorOf*.
 2. Given a relation *edge* for a graph, infer a relation *path*.
 3. Given a DAG representing possible moves in a game,

The digraph shows the possible moves:



The person who makes the last (legal) move wins.

A position is *winning* if, when it's your turn to move and you're in that position, you can win — *if you play cleverly*.

If the graph is acyclic, that determines every position as *winning* or *¬winning*.

3-valued logics: A *partial interpretation* is a set of *ground literals*:

- *positive literals*: $r(\vec{a})$
- *negative literals*: $\neg r(\vec{a})$

(for r a k -ary relation scheme and \vec{a} a k -tuple of attribute values).

When we talk about *ground literals*, we identify $\neg\neg r(\vec{a})$ with $r(\vec{a})$ (just for convenience of language).

For I a partial interpretation and λ a ground literal,

- λ is **true** (t) in I if $\lambda \in I$;
- λ is **false** (f) in I if $\neg\lambda \in I$; and
- λ is **undefined** (?) in I otherwise.

Truth Tables: (with $*$ a wild-card for “anything”)

α	$\neg\alpha$
t	f
?	?
f	f

α	β	$\alpha \wedge \beta$
t	t	t
t	?	?
t	f	f
?	t	?
?	?	?
?	f	?
f	*	f

α	β	$\alpha \leftarrow \beta$
t	*	t
?	t	f
?	?	t
?	f	t
f	t	f
f	?	f
f	f	t

And the fixed-point condition:

Distinguish 2-kinds of cycles:

(pure) positive cycles:

$$\{ances(a, b) \leftarrow ances(a, c), ances(c, b)\}$$

$$\{husb(a, b) \leftarrow wife(b, a), wife(b, a) \leftarrow husb(a, b)\}$$

Of course, real examples often aren't that obvious.

(partly) negative cycles:

$$\{win(x) \leftarrow move(x, y), \neg win(y)\}$$

$$\{win(x) \leftarrow \neg lose(x), lose(x) \leftarrow \neg win(x)\}$$

$$\{a \leftarrow \neg b, b \leftarrow c, c \leftarrow d, c \leftarrow \neg a\}$$

It turns out that only negative cycles provide nasty examples. And, as in the final example above, it matters whether the recursion is through an even or odd number of \neg signs.

Cycle Detection via the Gelfond-Lifschitz transform:

1. **Given:** (a) grounded IDB \mathcal{G}
(b) a candidate interpretation \mathcal{D} for \mathcal{G} .
2. **In each rule:** grounded rule $a \leftarrow b_1, \dots, b_k, \neg c_1, \dots, \neg c_m$, evaluate all the *negative subgoals* in \mathcal{D} and “simplify”:
 - delete rules with subgoals $\neg c_i$ where c_i is true in \mathcal{D} ;
 - delete subgoals $\neg c_j$ where $\neg c_j$ is false in \mathcal{D} .

The result is called the *Gelfond-Lifschitz* transform of \mathcal{G} w.r.t. \mathcal{D} . Denote it $\mathcal{G}_{\mathcal{D}}$.

3. **Observe** that $\mathcal{G}_{\mathcal{D}}$ has no negative subgoals — it is “Horn” or “definite.” So it has a minimum model — call it $Min(\mathcal{G}_{\mathcal{D}})$.
4. **If** $I = Min(\mathcal{G}_{\mathcal{D}})$, we say \mathcal{D} is a *stable extension* of the EDB under the IDB.

Unfortunate features for database theorists:

1. There may be 0, 1, or many stable extensions of an EDB under an IDB.
2. Even for a fixed IDB and variable EDB’s, determining whether there is a stable extension is NP complete (in the size of the EDB).

Well-founded inference: Allow 3-valued logic for the intensional relations.

In determining the minimum model $Min(\mathcal{G}_I)$, choose the minimum possible truth value for each ground atom, with $f <? < t$.

Disadvantage of 3-valued approaches: Don't allow for reasoning by cases, as in

$$a : - \neg b$$

$$b : - \neg a$$

$$c : - a$$

$$c : - b$$

Example: We are given a table of airplane flights:

Airline	Flight	From	To	Departs	Arrives
---------	--------	------	----	---------	---------

Choose a flight that will

- Allow me, by continuing on other flights, to get from *Rabbit Hash* to *Gackle*
- *by midnight tonight*
- allowing at least *1 hour for each layover*
- and, if possible, traveling on only 1 airline.

Expressing Constraints:

← bad condition.

in pure DATALOG this just yields a contradiction when bad information is added.

with stable extensions we sometimes have the constraint failing in one extension but not in others.

in the well-founded case there's an extension with minimal information (the “well-founded partial model”), and if the constraint is violated there it's violated in all stable extensions.

Top-down vs. Bottom-up evaluation and recursive query optimization.

All inference engines involve backtracking.

Prolog does a backtracking search for proofs, and is *top-down* (a.k.a., *goal-directed*).

Benefit: Avoid making calculations you don't care about.

Cost: Make lots of the same calculations over & over again, and potentially get lost in circular recursions.

(Aside: XSB Prolog implements well-founded negation.)

Satisfiability solvers and related products are generally *bottom-up*: do backtracking searches for satisfying truth assignments.

Optimization Problems with grounding:

1. Grounding the rules blows up the space requirement.

Can partially alleviate the problem by adding *domain predicates* for grounding.

2. Grounding may make us separately reason out many individual cases of a single general inference.

(You Prologgers compare this to “most general unifiers.”)

Memoization & Magic Sets: Try to mix benefits of bottom-up and top-down.

Comments on Distributed Databases:

- **Scenario:** Data is spread out across many different computers.
- **And you can't answer a query** by copying all relevant data to one site — because
 - Data is private and you don't want the query answerer to learn it — e.g. corporate secrets, government secrets, personal records — or
 - You are afraid someone else will intercept the data (similar privacy concerns) — or
 - You don't have the time or computational power to transmit it — e.g., you're transmitting from a small mobile device.

¿ **Can you answer the query by passing small messages back and forth among databases?**

Different Partition Scenarios:

- **Vertical partition:** Everybody knows the database scheme. Some relations are stored on each machine.
Extreme privacy case: The only attribute values that may be shared are those that are in common relations.
- **Horizontal partition:** Similar, but generally some tuples from each relation are stored on each machine.
- **AI topic:** The database schemes don't quite match, so we must infer some tuples.

Results for vertical partitions:

Relational algebra (with no aggregates): So long as the query answer concerns only tuples drawn from relations on the machine to which you pose the query, the query may be answered by fixed finite number of message passes.

Example: *Pairs of students (s_1, s_2) where there is some professor they are both taking classes from this quarter.* Say our relation **register** is on one machine and **schedule** is on another.

DATALOG: Not all queries can be answered, even when they involve only attributes from common relations.