CS345 Notes for Lecture 11/27/96

CORAL

Developed by Raghu Ramakrishnan at Wisconsin. Similar in many respects to NAIL/GLUE, but:

- All modules are declarative (no GLUE-style control flow).
- Programmer can declare the approach to be used to evaluate the predicates of a module. Strategies include:
  - a) Seminaive + magic-sets.
  - b) Seminaive + tail-recursion elimination as in Ch. 15 of PDKS-II.
  - c) Top-down, using Prolog-like algorithm.
- Built on top of Exodus data manager. Supports transactions, but like NAIL it is really a main-memory system.
- Store tuples with variables. Allows facts like $anc(X, X)$ = "everyone is his/her own ancestor." Also gives a magic-sets algorithm that avoids the need for subgoal rectification (see Sect. 13.6 of PDKS-II).
- C++ interface.
  1. C++ extended with classes for relations.
  2. Subgoals in CORAL can be C++ procedures.

Negation in CORAL

- Supports modularly stratified negation and takes advantage of magic-sets.
- Trick: work on lowest module possible at all times.
- A rule for a magic predicate belongs to the module of the rule that spawned the magic rule (by the Group I construction).

- Above rule OK, even though there may be inter-module recursions that result, because we will judiciously order the firing of rules.

Example: Here are nonrecursive rules that become recursive when magic-sets is applied.

```
q(X,Y) :- q(X,Z) & p(Z,Y)
p(X,Y) :- a(X,Y)
```

Query: q^{bf}.

- If we use simplified magic-sets and skip the supplementary predicate, we get

```
m_p(Z) :- m_q(X) & q(X,Z)
q(X,Y) :- m_q(X) & q(X,Z) & p(Z,Y)
p(X,Y) :- m_p(X) & a(X,Y)
```

- There is a recursive cycle

$$m\_p \to p \to q \to m\_p$$

Handling Negation + Magic-Sets

- Simple example — (just plain) stratified.

```
only1(X,Y) :- arc(X,Y) &
        not path2(X,Y)
path2(X,Y) :- arc(X,Z) & arc(Z,Y)
```

- $only1(X,Y)$ means there is an arc from node X to Y, but no path of length 2 from X to Y.

- Stratum 0 = $\{arc, path2\}$; Stratum 1 = $\{only1\}$.

- Query: $only1^{ff}$.

- Note, m_only1 has 0 arguments; in effect the initializer is `m_only1() :- true`. We can just drop m_only1 instead.

```
r1: m_path2(X,Y) :- arc(X,Y)
r2: path2(X,Y) :- m_path2(X,Y) &
        arc(X,Z) & arc(Z,Y)
r3: only1(X,Y) :- m_path2(X,Y) &
        not path2(X,Y)
```

- Suppose EDB = $\{1 \to 2,\ 2 \to 3,\ 1 \to 3\}$.

- In seminaive evaluation, we would infer $only1(1,3)$ on the first round, before we discover $path2(1,3)$ and thus falsify $\neg path2(1,3)$. I.e., we get the wrong answer.

A Technique for Modularly Stratified Negation

If the original rules are modularly stratified, the following approach works.

1. "Fire" the rules of one module at a time. Always pick a module such that all the modules below it have not inferred anything new the last time they were fired.

2. Group a rule for m_p with the module for predicate q if that magic rule came from passing queries from a rule for q to some subgoal with predicate p.

- Note that rules for m_p can thereby be partitioned into several modules.

Example: In the example above, r_2 is in the lower module ($path2$) and r_1 and r_3 are in the upper module ($only1$).

- When we ask a question like $m\_path2(1,2)$, we fire all possible attempts to answer this query in the module below before concluding $path2(1,2)$ is false.

- We don't return to the upper module until all attempts have failed. At that time, it is safe to conclude $\neg path2(1,2)$, and we infer $only1(1,2)$ by r_3.

Validity

Validity is a next-generation DBMS being developed by Machines Bull.

- Its query language is called DEL (Datalog Extended Language).

DEL

The language is "object-oriented Datalog." The important object-oriented extensions:

1. Tuples have object-ID's.

   - ☐ One can reference tuples by a *reference variable*, e.g., `p -> Name` for the `Name` component of the tuple referenced by `p`.

2. There is a conventional OO type system, including types that are lists, sets, bags, or arrays of other types.

3. There is inheritance by subclasses.

4. Classes can have methods.

5. Atoms are written with their arguments as "attribute : value."

Example:

```
epo(name: 'sally',
    phone: P,
    office: O)
```

- An argument whose value is a variable appearing only once in a rule need not appear.

Language Structure

- Similar to GLUE/NAIL: an imperative component and a declarative component.

- Declarative rules look like Datalog, but with the style of atoms indicated above, `<-` instead of `:-`, and logical connectives as words, e.g., `AND`.

Example:

```
path(first: X, last: Y) <-
    arc(tail: X, head: Z) AND
    arc(tail: Z, head: Y)
```

- Imperative statements include insert, delete, if, switch, create (for classes/predicates), new (for creating new objects), and the foreach iterator construct, illustrated below.

Example: Suppose we have a class `emp` with attributes that include `salary` of type integer and `mgr` of type `ref(emp)` (i.e., a reference to an employee object).

- The following deletes all employees who make more than their managers.

```
(1) FOREACH e:ref(emp) IN SET OF {X}
(2)     FROM X:emp(salary:S, mgr:m) DO
(3)         IF(S > m->salary) DELETE e;
```

- Line (2) declares `X` to range over the `emp` objects that have a salary and manager attribute (if there are subclasses, perhaps some do not). It also gives names `S` to the salary and `m` to the (reference to the) manager.

- Line (1) declares `e` to be a pointer (reference) ranging over all the tuples in the set constructed in line (2).

- Line (3) tests whether the salary of the employee is greater than the salary of the manager employee pointed to by `m`, and deletes the employee object if so.