Cours 2 - MIF14
Bases de données déductives

Datalog

Logical Rules

Recursion

Logic As a Query Language

- If-then logical rules have been used in many systems.
  - Important example: EII (Enterprise Information Integration).
- Nonrecursive rules are equivalent to the core relational algebra.
- Recursive rules extend relational algebra and appear in SQL-99.

Example: Enterprise Integration

- Goal: integrated view of the menus at many bars Sells(bar, beer, price).
- Joe has data JoeMenu(beer, price).
- Approach 1: Describe Sells in terms of JoeMenu and other local data sources.

Sells('Joe''s Bar', b, p) <- JoeMenu(b, p)

EII – (2)

- Approach 2: Describe how JoeMenu can be used as a view to help answer queries about Sells and other relations.

JoeMenu(b, p) <- Sells('Joe''s Bar', b, p)

- More about information integration later.

A Logical Rule

- Our first example of a rule uses the relations Frequents(drinker, bar), Likes(drinker, beer), and Sells(bar, beer, price).
- The rule is a query asking for "happy" drinkers --- those that frequent a bar that serves a beer that they like.

Anatomy of a Rule

Happy(d) <- Frequents(d,bar) AND
Likes(d,beer) AND Sells(bar,beer,p)

Head = consequent, a single subgoal

Read this symbol "if"

Body = antecedent = AND of *subgoals*.

Subgoals Are Atoms

- An *atom* is a *predicate*, or relation name with variables or constants as arguments.
- The head is an atom; the body is the AND of one or more atoms.
- Convention: Predicates begin with a capital, variables begin with lower-case.

Example: Atom

Sells(bar, beer, p)

The predicate = name of a relation

Arguments are variables (or constants).

Interpreting Rules

- A variable appearing in the head is *distinguished* ; otherwise it is *nondistinguished*.
- Rule meaning: The head is true for given values of the distinguished variables if there exist values of the nondistinguished variables that make all subgoals of the body true.

Example: Interpretation

Happy(d) <- Frequents(d,bar) AND
    Likes(d,beer) AND Sells(bar,beer,p)

Distinguished variable

Nondistinguished variables

Interpretation: drinker *d* is happy if there exist a bar, a beer, and a price *p* such that *d* frequents the bar, likes the beer, and the bar sells the beer at price *p*.

Applying a Rule

- Approach 1: consider all combinations of values of the variables.
- If all subgoals are true, then evaluate the head.
- The resulting head is a tuple in the result.

Example: Rule Evaluation

```
Happy(d) <- Frequents(d,bar) AND
        Likes(d,beer) AND Sells(bar,beer,p)
FOR (each d, bar, beer, p)
  IF (Frequents(d,bar), Likes(d,beer), and
  Sells(bar,beer,p) are all true)
        add Happy(d) to the result
```

- Note: set semantics so add only once.

A Glitch (Fixed Later)

- Relations are finite sets.
- We want rule evaluations to be finite and lead to finite results.
- "Unsafe" rules like P(x)<-Q(y) have infinite results, even if *Q* is finite.

Arithmetic Subgoals

- In addition to relations as predicates, a predicate for a subgoal of the body can be an arithmetic comparison.
- We write arithmetic subgoals in the usual way, e.g., $x < y$.

Example: Arithmetic

- A beer is "cheap" if there are at least two bars that sell it for under $2.

```
Cheap(beer) <- Sells(bar1,beer,p1) AND
      Sells(bar2,beer,p2) AND p1 < 2.00
      AND p2 < 2.00 AND bar1 <> bar2
```

Negated Subgoals

- NOT in front of a subgoal negates its meaning.
- Example: Think of Arc(a,b) as arcs in a graph.
  - S(x,y) says the graph is not transitive from *x* to *y* ; i.e., there is a path of length 2 from *x* to *y*, but no arc from *x* to *y*.

S(x,y) <- Arc(x,z) AND Arc(z,y)
AND NOT Arc(x,y)

Unsafe Rules

- Each of the following is unsafe and not allowed:
  1. P(x)<-Q(y)
  2. S(x) <- R(y) AND NOT R(x)
  3. S(x) <- R(y) AND x < y
  4. S(X, Y) :– X > Y
- In each case, an infinity of *x* 's can satisfy the rule, even if *R* is a finite relation.

Safe Rules

- A rule is *safe* if:
  1. Each distinguished variable (in the head),
  2. Each variable in an arithmetic subgoal, and
  3. Each variable in a negated subgoal,

  also appears in a nonnegated, relational subgoal of the body.
- Safe rules prevent infinite results.

Datalog Programs

- *Datalog program* = collection of rules.
- In a program, predicates can be either
  1. EDB = *Extensional Database* = stored table.
  2. IDB = *Intensional Database* = relation defined by rules.
- Never both! No EDB in heads.

Evaluating Datalog Programs

- As long as there is no recursion, we can pick an order to evaluate the IDB predicates, so that all the predicates in the body of its rules have already been evaluated.
- If an IDB predicate has more than one rule, each rule contributes tuples to its relation.

Example: Datalog Program

- Using EDB Sells(bar, beer, price) and Beers(name, manf), find the manufacturers of beers Joe doesn't sell.

```
JoeSells(b) <- Sells('Joe''s Bar', b, p)
Answer(m) <- Beers(b,m)
                    AND NOT JoeSells(b)
```

Example: Evaluation

- Step 1: Examine all Sells tuples with first component ’Joe’’s Bar’.
  - Add the second component to JoeSells.
- Step 2: Examine all Beers tuples (b,m).
  - If *b* is not in JoeSells, add *m* to Answer.

Expressive Power of Datalog

- Without recursion, Datalog can express all and only the queries of core relational algebra.
  - The same as SQL select-from-where, without aggregation and grouping.
- But with recursion, Datalog can express more than these languages.
- Yet still not Turing-complete.

Recursive Example

- EDB: Par(c,p) = *p* is a parent of *c*.
- Generalized cousins: people with common ancestors one or more generations back:

```
Sib(x,y) <- Par(x,p) AND Par(y,p) AND x<>y
Cousin(x,y) <- Sib(x,y)
Cousin(x,y) <- Par(x,xp) AND Par(y,yp)
                 AND Cousin(xp,yp)
```

Definition of Recursion

- Form a *dependency graph* whose nodes = IDB predicates.
- Arc *X* -> *Y* if and only if there is a rule with *X* in the head and *Y* in the body.
- Cycle = recursion; no cycle = no recursion.

Example: Dependency Graphs

$$\begin{tikzcd}\text{Cousin} \arrow[loop right] \arrow[d] \\ \text{Sib}\end{tikzcd}$$

Recursive

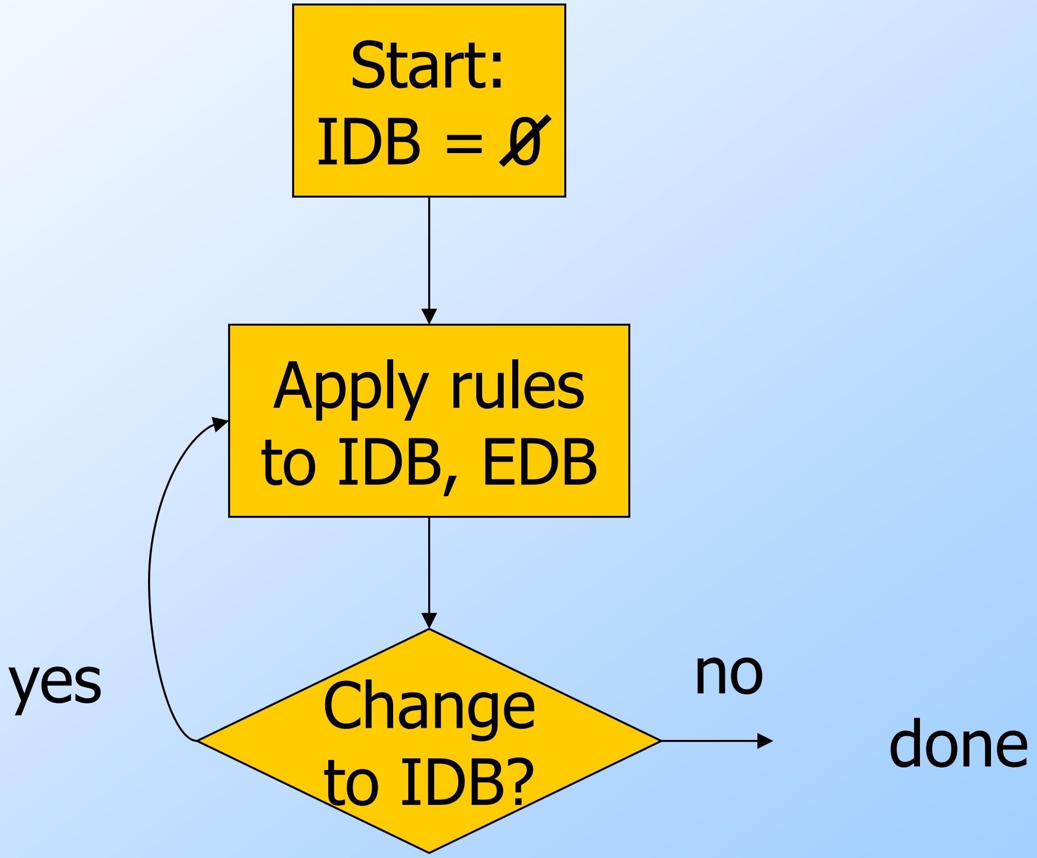
$$\begin{tikzcd}\text{Answer} \arrow[d] \\ \text{JoeSells}\end{tikzcd}$$

Nonrecursive

Evaluating Recursive Rules

- The following works when there is no negation:
  1. Start by assuming all IDB relations are empty.
  2. Repeatedly evaluate the rules using the EDB and the previous IDB, to get a new IDB.
  3. End when no change to IDB.

The “Naïve” Evaluation Algorithm


Start:
IDB = Ø
Apply rules
to IDB, EDB
Change
to IDB?
yes
no
done