Magic Sets

- Optimization technique for recursive Datalog.
- Also a win on some nonrecursive SQL (Mumick, Finkelstein, Pirahesh, and Ramakrishnan, 1990 SIGMOD, pp. 247–258).
- Combines benefits of both top-down (backward chaining, recursive tree search) and bottom-up (forward chaining, naive, seminaive) processing of logic, without disadvantages of either.

Example of Nonrecursive Use

Find the programmers who are making less than the average salary for their department.

```
SELECT e1.name
FROM Emps e1
WHERE e1.job = 'programmer' AND
    e1.sal < (
        SELECT AVG(e2.sal)
        FROM Emps e2
        WHERE e2.dept = e1.dept
    );
```

- Naive implementation computes the average salary for all departments.
- "Magic-sets" implementation first determines the departments that have programmers (perhaps very few). It can then use an index on `Emps.dept` to avoid accessing the entire `Emps` relation.

Recursive Example

```
anc(X,Y) :- par(X,Y)
anc(X,Y) :- par(X,Z) & anc(Z,Y)
```

- Query: $anc(0, W)$.
- Top-down search (e.g., Prolog) would:
  1. Query the EDB for $par(0, Y)$.
  2. By the first rule: return all such answers, say $\{(0, 1),\ (0, 2)\}$.
  3. The same parent facts are also useful in the second rule to set up "calls" to $anc(1, Y)$ and $anc(2, Y)$.
  4. Recursively solve these queries.

Advantage of Top-Down

- We never even ask about individuals that are not in the ancestry of individual 0.

Advantage of Bottom-Up

(i.e., naive, seminaive)

- We don't go into infinite recursive loops.

Example

Both of the following Datalog programs loop if evaluated top-down:

```
anc(X,Y) :- par(X,Y)
anc(X,Y) :- anc(X,Z) & par(Z,Y)

anc(X,Y) :- par(X,Y)
anc(X,Y) :- anc(X,Z) & anc(Z,Y)
```

Key Magic-Sets Ideas

1. Introduce "magic predicates" to represent the bound arguments in queries that a top-down search would ask.
2. Introduce "supplementary predicates" to represent how answers are passed from left-to-right through a rule.
3. Technical details to get right:
   a) *Predicate splitting*: an IDB predicate must be "called" (in top-down search) with only one binding pattern.
   b) *Subgoal rectification*: avoid IDB subgoals with repeated variables.

Rule/Goal Graphs

- Needed to assure unique binding patterns for IDB predicates.
- Composed of *rule* and *goal nodes*, as follows.

Goal Nodes

- Predicate + "adornment."
- *Adornment* = list of b's and f's, indicating which arguments are bound, which are free.
- Example: p^{bfb}. First and third arguments of p are bound.

Rule Nodes

- $r_i^{[S|T]}$ represents the point in rule r after seeing i subgoals, with variables in set S bound, those in T free.

Children of Goal Nodes

Children of goal node p^α are those rule nodes $r_0^{[S|T]}$ such that

1. Rule r has head predicate p.
2. S is the set of variables that appear in those arguments of the head that α says are bound.
3. T is the other variables of r.

Children of Rule Nodes

Children of the rule node $r_j^{[S|T]}$ are:

1. The goal node of the $(j+1)$st subgoal of r, with adornment that binds those arguments whose only variables are in S.
2. The rule node $r_{j+1}^{[S'|T']}$, where $S' = S +$ variables appearing the in $(j+1)$st subgoal; T' is the other variables.

- Exceptions: no r_{j+1} rule node if r has only $j+1$ subgoals. No goal child if $j = 0$ and r has no subgoals.

Constructing the RGG

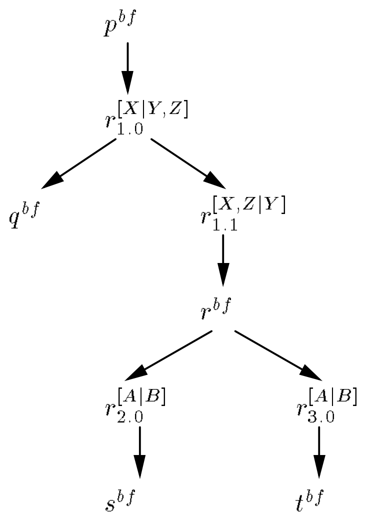
- Start with goal node whose adornment matches bindings of query.
- Add nodes by constructing children as required by rules from previous slides.
- Reordering of subgoals of a rule is allowed: helps maximize "bound" arguments.
- Reordering may be different for different rule nodes.

Example

Here is a nonrecursive example, where the RGG is a tree.

r_1: `p(X,Y) :- q(X,Z) & r(Z,Y)`
r_2: `r(A,B) :- s(A,B)`
r_3: `r(A,B) :- t(A,B)`

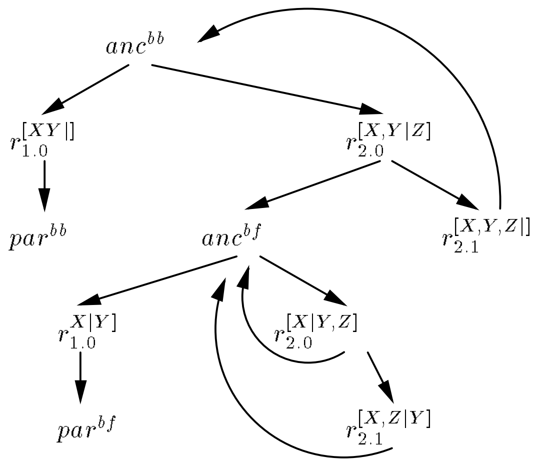
- Query form p^{bf}, e.g., $p(0,W)$?



Recursive Example

```
r1: anc(X,Y) :- par(X,Y)
r2: anc(X,Y) :- anc(X,Z) & anc(Z,Y)
```

- Query; anc^{bb}, e.g., $anc(joe, sue)$?



Splitting Predicates

- For magic-sets to work, there must be a unique binding pattern associated with each IDB predicate.

- No constraint on EDB predicates.

- Key idea: For each adornment α such that p^α appears in the RGG, make a new predicate

p_α. Rules for p_α are the same as for p, but predicates of IDB subgoals are the version with the correct binding pattern.

- RGG helps us figure out the needed binding patterns.

Example

For RGG above:

```
anc_bb(X,Y) :- par(X,Y)
anc_bb(X,Y) :- anc_bf(X,Z) &
        anc_bb(Z,Y)

anc_bf(X,Y) :- par(X,Y)
anc_bf(X,Y) :- anc_bf(X,Z) &
        anc_bf(Z,Y)
```

Rectifying Subgoals

- All IDB subgoals must have arguments that are distinct variables.
- Feasible for datalog (no function symbols).
- Fixes some problems where RGG knows about fewer bound arguments than the top-down expansion does.
  - ❖ See p. 801ff of PDKS-II.
- Trick: replace an IDB subgoal G with variables appearing in more than one argument and/or constant arguments by a new predicate whose arguments are single copies of the variables appearing in G.
- Create rules for the new predicate by unifying G with heads of rules for G's predicate.
- Repetition may be needed because the resulting rules may have unrectified subgoals.

Example

r_1: `p(X,Y) :- a(X,Y)`
r_2: `p(X,Y) :- b(X,Z) & p(Z,Z) & b(Z,Y)`

- $p(Z,Z)$ is unrectified. Create $q(Z) = p(Z,Z)$.
- Unify heads of rules with $p(Z,Z)$. Careful! Z in body of r_2 must be renamed.
- r_1 becomes `p(Z,Z) :- a(Z,Z)` or

  `q(Z) :- a(Z,Z)`

- r_2 becomes

  `p(Z,Z) :- b(Z,W) & p(W,W) & b(W,Z)`

  or `q(Z) :- b(Z,W) & q(W) & b(W,Z)`

- Finally, in the original r_2 we replace subgoal $p(Z,Z)$ by $q(Z)$. The resulting rules, with variables renamed:

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

  q(X) :- a(X,X)
  q(X) :- b(X,Y) & q(Y) & b(Y,X)
  ```

Magic Sets Transformation

Start with a program and a binding pattern for a query.

1. Split predicates to get unique binding patterns.
2. Rectify subgoals.
3. Introduce magic and supplementary predicates as follows.

Magic Predicates

For each IDB predicate p, introduce m_p.

- Arguments of m_p correspond to bound arguments of p in its unique binding pattern.
- Intuition: m_p is true of exactly those tuples that are members of queries to some p-node in the top-down expansion.

Supplementary Predicates

For each rule r of n subgoals, introduce supplementary predicates $sup_{r.j}$ for $0 \leq j < n$.

- Arguments are the bound and *active* variables before the $j+1$st subgoal of r.
  - ❖ A variable is active iff it appears either in the head or a subgoal from $j+1$ on.
- Intuition: true for a tuple iff that tuple represents a possible binding for the bound, active variables at that point.